PH.D. DISSERTATION

ICT INTERNATIONAL DOCTORAL SCHOOL
DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE
UNIVERSITY OF TRENTO

Optimization Modulo Theories with OPTIMATHSAT

PATRICK TRENTIN

Advisor:
Prof. Roberto Sebastiani
DISI, UNIVERSITY OF TRENTO, ITALY

MAY 2019

Abstract

In the contexts of Formal Verification (FV) and Automated Reasoning (AR), Satisfiability Modulo Theories (SMT) is an important discipline that allows for dealing with industrial-level decision problems. Optimization Modulo Theories (OMT) extends Satisfiability Modulo Theories with the ability to express, and optimize, objective functions.

Recently, there has been a growing interest towards OMT, as witnessed by an increasing number of applications using, at their core, some OMT solver as main power-horse engine. However, at present few OMT solvers exist, and the development of OMT technology is still at an early stage, with large margins of improvement. We identify two major advancement directions in particular. First, there is a general need for closing the expressiveness gap with respect to SMT, and provide optimization procedures that can deal with the wider range of theories supported by SMT solvers. Second, there is an urgent need for more efficient techniques that can improve on the performance of state-of-the-art OMT solvers, because solving an OMT problem is inherently more expensive than dealing with its SMT counterpart, often by at least one order of magnitude.

In this dissertation, we present a variety of techniques that deal with the identified issues and advance both the expressiveness and the efficiency of OMT. We describe our implementation of these techniques inside OPTIMATHSAT*, a state-of-the-art OMT solver based on* MATHSAT5*, along with its high-level architecture, Input/Output interfaces and configurable options. Thanks to our novel contributions,* OPTIMATHSAT *can now deal with the single- and the multi-objective incremental optimization of goals defined over multiple domains –the Boolean, the mixed Linear Integer and Rational Arithmetic, the Bit-Vector and the Floating-Point domain– including (Partial Weighted)* MAXSMT*.*

We validate our theoretical contributions experimentally, by comparing the performance of OPTIMATHSAT *against other, competing, OMT solvers. Finally, we investigate the effectiveness of OMT beyond the scope of Formal Verification, and describe an experimental evaluation comparing* OPTIMATHSAT *with Finite Domain Constraint Programming tools on benchmark-sets coming from their respective domains.*

Keywords

[Satisfiability Modulo Theories; SMT; Optimization Modulo Theories; OMT; OptiMathSAT;

Partial Weighted MaxSMT; Constraint Programming; Multi-Objective Optimization; MiniZinc]

Acknowledgements

A doctoral thesis is often described as a solitary endeavour; however the long list that follows definitely proves the opposite.

First, I would like to express my sincere gratitude to my advisor Prof. Roberto Sebastiani for the continuous support of my Ph.D. study, for his patience, motivation and brilliant insights. His guidance helped me throughout my entire research and also with writing this thesis. I could not have imagined having a better advisor and mentor for my Ph.D. study. Besides my advisor, I would also like to thank the rest of my thesis committee: Prof. Albert Oliveras, Prof. Luigi Palopoli and Prof. Cesare Tinelli, for taking the time to review this thesis and for their insightful feedback on its contents. After spending so many years at this institution, I feel that I also owe my gratitude to the staff of the ICT Doctoral School and the DISI department of the University of Trento. In particular, I would like to thank the two members of the doctoral school staff, Francesca Belton and Andrea Stenico, and the four members of the technical staff, Alessandro Tomasi, Mario Passamani, Veronica Rizzi and Danilo Severina. All of them have demonstrated, time after time, great dedication to their job as well as a rare inclination for kindness and willingness to help. A special thank goes to Michela Angeli, a former employee of this institution, whose work lifted me from the burden of dealing with the task of organizing my work-related travels and helped me focusing on my research at the early stages of this Ph.D. study.

Second, I would like to thank Dr. Silvia Tomasi who, together with Prof. Roberto Sebastiani, laid the foundations for this research and built the first version of OPTIMATHSAT, before advancing her career elsewhere and passing on the torch to me. I would also like to thank the members of the MATHSAT team, namely Dr. Alberto Griggio, Dr. Alessandro Cimatti and Prof. Roberto Sebastiani, as well as the countless people who gave their contribution to MATHSAT over the years: a significant portion of OPTIMATHSAT's success is the consequence of their efforts. I am especially grateful to Dr. Alberto Griggio who has helped me with MATHSAT5 code and has always shown both an unlimited amount of patience and a gracious sense of humor in all of our interactions, despite the sharp expertise gap among us.

Third, I would like to thank Prof. David Monniaux, who provided me with an opportunity to join his team at VERIMAG as an intern during my Ph.D. study, and provided me with both valuable guidance and feedback regarding this research work and on my research attitude in general.

Fourth, I would like to express my gratitude towards those people who have chosen to use OPTIMATHSAT for their own applications, investing their own time and efforts in the process. By doing so, they have given a meaning to my own work. I am especially grateful towards both

Dr. Mai Chi Nguyen and Dr. Stefano Teso, whose research provided useful benchmark-sets for OPTIMATHSAT and also helped steering my research efforts towards a fruitful direction. I hold the same amount of gratitude towards those bachelor's degree students (Luca Gasparetto) and master's degree students (Filippo Bigarella and Francesco Contaldo) who have seen the potential of this research and decided to cooperate with our research group during their studies. Working with everyone of them as been an invaluable chance of personal growth for me. I am especially grateful towards Francesco Contaldo for his tireless work on the OMT2MZN project, and also for allowing me to include part of his experimental results in this dissertation.

Fifth, I would like to thank the countless number of people working in my research domain, and those fields closely related to it, for sharing their insightful ideas with the community and helping me to expand my incredibly limited horizons. This thesis builds upon the ideas of many people and I hope that it can contribute to their success by spreading their work even further. I am also grateful towards those researchers and software maintainers who have kindly answered my questions and addressed my bug reports over the past years. Their help often came in when it was needed the most and in exchange for no more than a simple "thank you".

Last but not the least, I thank my friends for sharing this voyage with me, both those who have always remained at my side over the years as well as those who are now gone forever but have left a piece of themselves in my heart. I would like to deeply thank my parents, Fausto Trentin and Helga Gasser, who have sacrificed pursuing their own life goals in order to support their children even at hardship, and my sisters, Silvia and Deborah Trentin, who have given me the opportunity to gauge at life from different perspectives. I would also like to express my deepest gratitude to my father and mother in law, 蔡崇條 and 吳美花, who have welcomed me in their family –without any preconception– and have graciously granted me the honor of taking their only daughter, 蔡亞芳, as my beloved wife for the rest of our days. My spouse is my most enthusiastic cheerleader, she is my best friend and she is an amazing wife. Without her cheerful mind, I would be a much grumpier person; without her love and support, I would be lost. I am grateful to my wife not just because she has given up her own life to be at my side, but because she has spent sleepless nights taking care and supporting me in the moments when I most needed it. At long last, her unbending love has given peace to my restless soul, so it only seems right that I dedicate this dissertation to her:

謝謝你給我的一切，我的妻子。

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Satisfiability Modulo Theories (SMT) denotes the problem of deciding the satisfiability of a first-order formula with respect to a combination of decidable first-order theories [BSST09]. The last fifteen years have witnessed the development of very efficient SMT solvers, most of which are based on the so-called lazy-SMT schema that combines the power of modern Conflict-Driven Clause-Learning (CDCL) SAT solvers [MSLM09] with the expressiveness of dedicated decision procedures ($\mathcal{T}$*-solvers*) for several first-order theories of practical interest such as linear arithmetic over the rationals ($\mathcal{LRA}$), the integers ($\mathcal{LIA}$) or their combination ($\mathcal{LIRA}$), non-linear arithmetic over the reals ($\mathcal{NLRA}$) or the integers ($\mathcal{NLIA}$), arrays ($\mathcal{AR}$), bit-vectors ($\mathcal{BV}$), floating-point arithmetic ($\mathcal{FP}$), and their combinations thereof. (See [NOT06, Seb07, BSST09] for an overview.). This has brought previously-intractable problems to the reach of state-of-the-art SMT solvers, so much so that SMT has acquired a prominent role in many applications of industrial interest such as Formal Verification (FV) of hardware and software systems, Automated Reasoning (AR), resource planning, temporal reasoning and scheduling of real-time embedded systems.

Optimization Modulo Theories (OMT), [NO06, CFG$^+$10, ST12, DDMA12, CGSS13a, MP13, LAK$^+$14, LORR14, ST15a], is a more recent extension to Satisfiability Modulo Theories that allows for finding a model of a first-order formula that is *optimal* with respect to some objective function through a combination of SMT and optimization procedures. Many SMT problems of practical interest are derived from —or can be easily extended to— an optimization problem. This is the case of SMT-based model checking with timed or hybrid systems, (e.g. [ACKS02, ABCS05]), in which one may want to find those executions that optimize the value of some parameter (e.g., a clock timeout value or the elapsed time) while full-filling or violating some temporal logic property. For instance, [ST15a] suggests using OMT to find the minimum opening time interval that causes a safety violation for a level crossing barrier.

A non-exhaustive list of the most recent OMT applications can be found towards the end of this dissertation, while a few examples of less-recent OMT applications can also be found in [NO06, CFG+10, ST12, CGSS13a, LAK+14, BP14, LORR14, ST15a, ST15c, ST17, ST18].

Since the seminal work of Nieuwenhuis and Oliveras in [NO06], which have dealt with the Optimization Modulo Theories problem for first, there has been an increasing interest around this topic. However, the research on OMT still appears to be at an early stage when compared with other, more thoroughly studied, fields such as SMT. In fact, the number of scientific publications in the literature that extend SMT with *optimization* is fairly limited, [NO06, CFG+10, Roc11, ST12, DDMA12, MP13, CGSS13a, LAK+14, LORR14, ST15a], and there are only a handful of more recent works that can be added to this figure, [BP14, BPF15, ST15b, ST15c, ABCF16, NR16, ST17, AAdB+17, ST18, FBB18, KBE18, AAdB+18, TS19], although we expect more to appear.

To this date, few OMT solvers exist that are also applicable beyond the scope of *Partial Weighted* MAXSMT, namely BCLT [NO06, BNO+08, Roc11, LORR14], CEGIO [ABCF16, AAdB+17, AAdB+18], HAZEL [NR16], OPTIMATHSAT [ST12, ST15a, ST15b, ST15c, ST17, ST18, TS19], PULI [KBE18], SYMBA [LAK+14] and Z3 [BP14, BPF15]. To this aim, we observe that most of these solvers focus on different, partially overlapping, niche subsets of Optimization Modulo Theories, and employ different standards for their Input/Output interfaces. In practice, the combination of these two factors makes it hard to compare with one another the few OMT solvers in existence.

In what follows we focus on the state of the art prior of the start of this Ph.D. program (November 1st, 2014). Then, we make the following observations. Some of these works, such as [NO06, CFG+10, CGSS13a], are applicable only to *Partial Weighted* MAXSMT, which is strictly less expressive than Optimization Modulo Theories with Linear Integer/Rational Arithmetic cost functions [ST15a]. The optimization procedures described by Sebastiani and Tomasi in [ST12, ST15a] and by Li et al. in [LAK+14] can only deal with OMT with a *Linear Rational Arithmetic* objective. In contrast, those described by R. O. Vendrel in [Roc11] and Manolios et al. in [MP13] deal with *Linear Integer Arithmetic* objectives only. Neither of these works expressively supports *Mixed Linear Integer and Rational Arithmetic* objectives, and [LAK+14] is the only work describing some form of multi-objective optimization support. Their approach, however, is limited to the case in which every goal can be considered a completely independent target from all the others, akin to solving a number of separate OMT problems in parallel. Last, we observe that none of these works on OMT describes an incremental interface allowing for pushing and popping subformulas and objective functions from the formula stack, allowing for reusing useful information from one call to the other. This appears to be a relevant limitation because SMT back-ends are often invocked incrementally like, e.g., in the context of Formal

Verification.

Based on the above considerations on the status of Optimization Modulo Theories before the start of this research, we identify the following improvements as necessary milestones to unleash the full potential of Optimization Modulo Theories and make it further adopted in the context of real-world, industrial-level, applications.

First, there is a general need to bridge the expressiveness gap of Optimization Modulo Theories when compared to Satisfiability Modulo Theories, and go beyond the various *Linear Arithmetic* restrictions that are currently supported by OMT and MAXSMT solvers. This not only means introducing optimization procedures that can handle *Mixed Linear Integer and Rational Arithmetic* goals and combination of theories at the same time, but also objective functions defined in other theories than *Linear Arithmetic*, such as the theory of Bit-Vectors and the theory of Floating-Point numbers.

Second, it is necessary to investigate possible approaches for improving the performance of OMT solvers, including methods that are only applicable to objective functions abiding to certain format restrictions (like, e.g., MAXSMT). This goal is easily justifiable by observing that while the leap from SMT to OMT can appear to be quite small from the perspective of an end-user, due to the ease with which a new objective can be both declared and used, in practice solving an OMT problem can be orders of magnitude harder in terms of computational effort than dealing with its SMT counter-part. This is due to the fact that in an SMT solver the search stops as soon as the first feasible solution is hit, whereas in an OMT solver the search is normally terminated only when an optimal solution is found. In practice, dealing with the latter goal is a significantly more daunting task than satisfiability, that can require enumerating multiple suboptimal solutions along the search and also certifying the absence of an improving solution at the end of the optimization search. Therefore, we believe that Optimization Modulo Theories can greatly benefit from the investigation of more efficient techniques that may bring previously intractable applications within the reach of OMT solvers.

A third improvement direction, closely related to the previous goal, is *incrementality*. This refers to the ability to push and pop both constraints and objectives on the formula stack of an OMT solver, while being able to reuse useful information automatically learned by the OMT solver across multiple optimization searches. This feature, that is widely supported among SMT solvers, can be crucial for achieving significant search speed-up when dealing with applications that require performing a large number of incremental calls to the tool, each bringing small changes to the previous formula.

Fourth, Optimization Modulo Theories can also greatly benefit from further extending the pioneering work of Li et al. on multi-objective optimization in [LAK+14]. In particular, some OMT applications —like, e.g., [NSGM16a, NSGM16b, NSGM17]— may benefit from the

ability of performing Lexicographic or Pareto-optimization search over a set of constraints.

Fifth, we believe it is worth considering whether the range of applications of Optimization Modulo Theories can be extended further beyond its natural Formal Verification (FV) and Automated Reasoning (AR) domains, and to what extent. In this respect, it is worth noting that there are other tool families that allow for optimizing some objective function under a set of constraints, such as *Finite Domain Constraint Programming* (FDCP) and *Mixed Integer Linear Programming* (MILP). Among its strengths, Optimization Modulo Theories can rely on native Boolean reasoning capabilities, that may prove to be an edge when dealing with highly combinatorial problems, and the ability to handle theories that are commonly not supported by tools coming from other research fields, such as the theory of arrays ($\mathcal{AR}$) or the theory of uninterpreted functions with equality ($\mathcal{EUF}$). Previous works, such as [BPV09, BPSV09, BSV10, BPSV12], have already started investigating those applications for which Satisfiability Modulo Theories can be competitive with FDCP and MILP tools. Part of the work described in this thesis is devoted to pushing forward this research work, focusing on Optimization Modulo Theories.

On the whole, the expected benefits from the rise of more efficient and expressive Optimization Modulo Theories solvers are many-fold.

- In those application fields that already benefit from SMT technology —like, e.g., Formal Verification and AI Planning— the rise of efficient OMT tools should allow for addressing more sophisticate versions of those problems (like, e.g., planning with resource optimization).
- In those application fields that typically make use of FDCP/MILP back-end engines instead —like, e.g., scheduling, industrial plant optimization [JG02], electrical grid optimization [TCHS13], planning with resources [BLPS15, DPSS15]— OMT could play as an interesting alternative, taking advantage from several distinctive features that can be inherited from SMT: the native (and very efficient) handling of Boolean operators, incrementality, the capability of combining arithmetic with other theories (e.g. $\mathcal{EUF}$, arrays, bit-vectors, data-structures, ...).
- Other application fields —e.g., machine learning, conceptual modeling— can benefit from OMT as enabling technology for new approaches. For example, the very recent Structured Learning Modulo Theories [TSP17] and goal-modeling optimization [cgm] already employ the OMT solver OPTIMATHSAT as back-end engine.

Contributions

In this dissertation, we broaden the horizon of Optimization Modulo Theories along several directions, including

- A combination of SMT, Linear Programming (LP) and Integer Linear Programming (ILP) techniques for dealing with OMT problems with *Mixed Linear Integer and Rational Arithmetic* objectives (§4.1);
- An effective use of sorting network circuits to speed up the optimization search when dealing with OMT problems with Pseudo-Boolean cost functions and *Partial Weighted* MAXSMT problems (§4.2.1);
- An extension of the OMT($\mathcal{LRA} \cup \mathcal{T}$) procedures presented in [ST12, ST15a] to deal with OMT problems with Bit-Vector (§4.3.1) and Floating-Point (§4.4.1) cost functions;
- A generalization of the maximization algorithms for unsigned Bit-Vectors presented in [NR16], that has not been co-authored by the Ph.D. candidate, to deal with the minimization or the maximization of both signed and unsigned Bit-Vector goals (§4.3.2 and §4.3.3);
- A novel algorithm based on binary-search for dealing with OMT problems with Floating-Point cost functions (§4.4.2), inspired by an analogous approach presented in [NR16] for handling unsigned $\mathcal{BV}$ goals;
- A description of two techniques for implementing, with little effort, an incremental OMT solver on top of an incremental SMT solver (§4.5);
- A definition of Multi-Objective Optimization Modulo Theories (§4.6), and all of its variants. An effective encoding for MINMAX and MAXMIN goals into single-objective OMT (§4.6.1). Optimization procedures for dealing with Multiple-Independent Optimization (§4.6.2), Lexicographic Optimization (§4.6.3) and Pareto Optimization (§4.6.4);

and more.

We have implemented all of these novel functionalities in **OPTIMATHSAT**, an OMT solver that has been first presented by Sebastiani and Tomasi in [ST12, ST15a] and that we have substantially re-implemented from scratch. OPTIMATHSAT is based on the MATHSAT5 SMT solver [mata, CGSS13b]. We describe the architecture of our new implementation in Chapter §5.

As part of the work subject of this dissertation, we have also incorporated into OPTIMATHSAT a few relevant OMT approaches that have been presented by other researchers working on

the same topics. This includes the *Maximum Resolution* engine for OMT with Pseudo-Boolean and MAXSMT objectives that has been first presented in [NB14, BP14], a modified version of the OBV-WA and OBV-BS algorithms for $\mathcal{BV}$ optimization that have been first presented in [NR16] and a porting from scratch of the lemma-lifting approach for MAXSMT problems that has been previously presented in [CGSS13a].

In addition, we have extended OPTIMATHSAT with three novel input/output interfaces, including

- an extended version of the SMT-LIBV2 format supporting the novel optimization functionalities (§5.3.1),
- a new MINIZINC interface for dealing with problems that are typically handled with FDCP solvers (§5.3.2), and
- a new API, with C, Python and Java bindings (§5.3.3).

We include, in Chapter §6, several experimental evaluations that we have performed to validate the approaches proposed in this thesis. Most experiments compare OPTIMATHSAT, using various configurations, against other state-of-the-art OMT solvers with similar solving capabilities, on benchmark-sets challenging the newly introduced functionalities. One of these experiments compares OPTIMATHSAT with a selection of *Finite Domain Constraint Programming* tools on formulas that are traditionally handled with FDCP solvers, and vice-versa (§6.6). Part of these results have been collected by a Master Degree student at University of Trento, Francesco Contaldo, under the co-supervision of the Ph.D. candidate, and they have been included in his Master Thesis together with a compiler from the SMT-LIBV2 format to MINIZINC, and also additional experimental data.

Publications

A significant part of the content of this dissertation has already been published in

- [ST15c] — Roberto Sebastiani and Patrick Trentin. Pushing the Envelope of Optimization Modulo Theories with Linear-Arithmetic Cost Functions. In *Proc. Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'15*, volume 9035 of *LNCS*. Springer, 2015
- [ST15b] — Roberto Sebastiani and Patrick Trentin. OptiMathSAT: A Tool for Optimization Modulo Theories. In *Proc. International Conference on Computer-Aided Verification, CAV 2015*, volume 9206 of *LNCS*. Springer, 2015

- [ST16] — Roberto Sebastiani and Patrick Trentin. On the Benefits of Enhancing Optimization Modulo Theories with Sorting Networks for MaxSMT. In *Proceedings of the 14th International Workshop on Satisfiability Modulo Theories, SMT-2016.*, CEUR Workshop Proceedings, 2016
- [ST17] — Roberto Sebastiani and Patrick Trentin. On Optimization Modulo Theories, MaxSMT and Sorting Networks. In *Proc. Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'17*, volume 10205 of *LNCS*. Springer, 2017
- [ST18] — Roberto Sebastiani and Patrick Trentin. OptiMathSAT: A Tool for Optimization Modulo Theories. *Journal of Automated Reasoning*, Dec 2018
- [TS19] — Patrick Trentin and Roberto Sebastiani. Optimization Modulo the Theory of Floating-Point Numbers. In *Proc. Int. Conference on Automated Deduction, CADE 27*, LNCS/LNAI. Springer, 2019. To appear.

Structure

This thesis is divided in two parts.

Part I provides an essential introduction to necessary background knowledge and terminology, and a survey of the related work. In particular,

Chapter §2 introduces the reader with essential notions about *Propositional Satisfiability* (§2.1) and *Satisfiability Modulo Theories* (§2.2), and then extensively covers the state of the art in *Optimization Modulo Theories* prior to the start of this Ph.D. (§2.3). We conclude this chapter with a bare-bone introduction to *Finite Domain Constraint Programming* and a survey of work in the literature across both fields that are of interest for this dissertation.

Chapter §3 provides an overview of the related work on *Optimization Modulo Theories* that is of interest for this dissertation. In particular, it surveys those scientific publications about OMT that have been published after the beginning of this Ph.D. study.

Part II is devoted to the description of the main contributions of this thesis. In detail,

Chapter §4 illustrates major advances in the context of Optimization Modulo Theories that occurred during the span of this Ph.D. study. Section §4.1 extends the OMT

procedures for *Linear Rational Arithmetic* objectives presented in [ST12, ST15a] to the case of *Mixed Integer Linear Arithmetic*. Section §4.2 deals with OMT problems wherein the cost function is a Pseudo-Boolean term, and also with *Partial Weighted* MAXSMT. In the first part, we show an effective use of sorting networks that enhances the basic optimization-search schema in use by the OMT solver. In the second part, we describe OPTIMATHSAT's implementation of the *Maximum Resolution* engine presented in [NB14, BP14]. Section §4.3 describes how to deal with Bit-Vector cost functions using a basic OMT-based search-schema, and also the OBV-WA and the OBV-BS algorithms first presented in [NR16]. Section §4.4 focuses instead on Floating-Point objective functions; it describes a basic OMT-based search-schema and also OFP-BS, a novel optimization search algorithm inspired by the OBV-BS engine in [NR16]. Section §4.5 illustrates two useful techniques for *incremental Optimization Modulo Theories* solving. Section §4.6 defines the *Multi-Objective Optimization* problem in the context of *Optimization Modulo Theories*. Then, it describes procedures for dealing with MINMAX/MAXMIN objectives, Multiple-Independent optimization, Lexicographic optimization and Pareto optimization. Section §4.7 concludes this chapter with a simple extension of *All-SMT* to the case of Optimization Modulo Theories.

Chapter §5 is devoted to the description of the implementation details concerning OPTIMATHSAT. Section §5.1 provides a high-level overview of its architecture, whereas Section §5.2 goes into the details of its distinctive approach to optimization, that currently sets it apart from other OMT solvers. Section §5.3 illustrates the Input/Output interfaces of OPTIMATHSAT, including its *Extended* SMT-LIBV2 *Interface*, its MINIZINC *Interface* and its public API. The Chapter is concluded with a detailed overview of the configurable options of OPTIMATHSAT, showing how to activate and use the functionalities described in Chapter §4 and their variants.

Chapter §6 presents a number of experimental evaluations that have been performed to validate, at an implementation level, the scientific contributions described in this thesis. Section §6.1 illustrates an experiment on OMT formulas with a *Linear Integer Arithmetic* objective. Section §6.2 deals with Pseudo-Boolean and *Partial Weighted* MAXSMT goals. Section §6.3 and Section §6.4 evaluate the procedures for Bit-Vector and Floating-Point optimization respectively. Section §6.5 compares basic single-objective optimization with incremental and multi-objective optimization. The Chapter is concluded in Section §6.6 with a comparison among OPTIMATHSAT and MINIZINC solvers on a few benchmark-sets coming from their respective domains.

Chapter §7 contains a survey of some relevant scientific works using Optimization Modulo Theories (§7.1) and the description of a handful of applications of OPTIMATHSAT that have been of primary importance in guiding the research and development of the features described in this dissertation (§7.2).

Finally, in **Chapter §8** we draw some conclusions on the research work presented in this thesis and we outline some possible directions for future development expanding on this research.

Part I

Background, State of the Art and Related Work

Chapter 2

Background & State of the Art

This chapter provides an introduction to background and state of the art concepts that are used throughout this dissertation. The presented material on *Propositional Satisfiability* and *Satisfiability Modulo Theories* is mostly taken from [NOT06, Seb07, BHvMW09, BSST09], whereas the content on *Optimization Modulo Theories* is based on a variety of publications on the topic, [NO06, CFG+10, Roc11, ST12, DDMA12, MP13, CGSS13a, LAK+14, LORR14, ST15a], and also on the background content of previous publications of the author of this dissertation [ST15b, ST15c, ST17, ST18, TS19].

The topics being presented are:

§2.1 *Propositional Satisfiability* (SAT): basic concepts and terminology, followed by an essential description of both DPLL and CDCL, the most well-known algorithms for deciding SAT.

§2.2 *Satisfiability Modulo Theories* (SMT): starts with basic concepts and terminology, and then focuses with a greater level of detail on the so-called "lazy" approach for SMT (§2.2.1). Various additional topics are touched, including incremental SMT (§2.2.2), some SMT theories of interest (§2.2.3) and combination of theories in SMT (§2.2.4).

§2.3 *Optimization Modulo Theories* (OMT): introduces the problem of performing optimization in SMT, also known as *Optimization Modulo Theories*. It covers OMT with *Linear Rational Arithmetic* cost functions (§2.3.1), OMT with *Linear Integer Arithmetic* cost functions (§2.3.2) and OMT with *Pseudo Boolean* and MaxSMT objectives (§2.3.3).

§2.4 *Constraint Programming and SAT/SMT/OMT*: introduces, in Section §2.4.1, *(Finite Domain) Constraint Programming* (FDCP), a restriction of *Constraint Programming* (CP) that is closely related to the domains of SAT, SMT and OMT solving. Then, after short overview of the MiniZinc and FlatZinc languages that are widely adopted by FDCP

solvers (§2.4.2), it reviews (§2.4.3) a few literature works crossing the border among the FDCP and the SAT, SMT and OMT domains, focusing only on those excerpts that we have deemed as being relevant for this dissertation.

Note. This chapter illustrates the state of the art in *Optimization Modulo Theories* at the time in which this Ph.D. was started (*November, 2014*). Therefore, with the exception of a few studies that were kept inside this chapter to preserve the continuity of presentation, any other OMT-related work which has been published after this date that we are aware of is described either in the Related Work (§3), among OMT Applications (§7.1) or among OPTIMATHSAT Applications (§7.2).

2.1 Propositional Satisfiability

Let $X \stackrel{\text{def}}{=} \{x_1, ..., x_n\}$ be a set of n Boolean variables. A *literal* l is a variable x_i or its negation $\neg x_i$. A *clause* C is a disjunction of literals $l_1 \vee ... \vee l_k$.

A propositional formula φ is said to be in *Conjunctive Normal Form* (CNF) if it is a conjunction of clauses $C_1 \wedge ... \wedge C_m$. Notation-wise, a clause C and a CNF formula φ are also often represented as a set of literals $\{l_1, ..., l_k\}$ and a set of clauses $\{C_1, ..., C_m\}$ respectively.

As outlined in [Tse68], for every non-CNF formula φ an equisatisfiable CNF formula φ' can be generated in polynomial time. Therefore, in the rest of this thesis we assume that every formula φ is in CNF.

In the context of propositional satisfiability (SAT) [BHvMW09], Boolean variables can be either *assigned* or *unassigned*. An assigned variable x evaluates to either $\bot$ (false) or $\top$ (true). When a variable x has not yet been assigned a value, we use $undef$ to denote its *undefined* value. An assignment is a function $\mu : X \rightarrow \{undef, \bot, \top\}$ that maps each assigned variable x to a value in $\{\bot, \top\}$ and the remaining variables to $undef$. An assignment μ is said to be a *complete (truth) assignment* if it maps all variables $x_i \in X$ to a value in $\{\bot, \top\}$. Otherwise, if there exists some $x_j \in X$ mapped to $undef$, it is a *partial (truth) assignment*. Given some μ, a clause C is said to be *satisfied* if at least one of these literals evaluates to $\top$, and *unsatisfied* otherwise. Given a (partial) truth assignment μ and a clause C, C is said to be a *unit clause* when all-but-one literals $l_i \in C$ evaluate to $\bot$, so that C can only be *satisfied* if μ assigns the remaining literal l_j to the value $\top$.

Given a CNF formula φ, the goal of *propositional satisfiability* is to find a (complete) truth assignment μ such that all clauses $C_i \in \varphi$ are satisfied. Formally, this is denoted by the writing $\mu \models \varphi$. In the following, we give a brief introduction to two of the most widely known

approaches for SAT. We refer the reader to the vast literature on propositional satisfiability for a more comprehensive introduction on the topic, e.g. [GKSS08, KBK09, LM09, MSLM09, Pre09, RM09].

Davis-Putnam-Logemann-Loveland (DPLL) Algorithm [DP60, DLL62]. Given a CNF formula φ, this procedure starts from an initially empty assignment μ, in which all variables x are undefined, and then tries to incrementally extend it to a complete truth assignment μ that propositionally satisfies φ.

The algorithm recursively selects some unassigned Boolean variable x_i belonging to some undecided clause C, and separately explores the case of assigning x to the value $\bot$ and to the value $\top$ (not necessarily in this order). The second case is explored only when the first one resulted in a contradiction, that is, it made some clause C unsatisfiable. Moreover, if both cases are found to be unsatisfiable, then the search procedure backtracks to the most recent point in which it can make a different, unexplored, decision on the value of some x_i (if any). The search terminates with SAT when the partial truth assignment μ is successfully extended to a complete truth assignment that propositionally satisfies φ. Otherwise, the procedure terminates with UNSAT when it unsuccessfully explored all possible cases.

A number of (satisfiability-preserving) clause-transformation *rules* are applied to greatly improve the efficiency of the DPLL procedure. Among these, a particularly important role is played by *unit propagation*. Given a unit clause C in which some $l_i \in C$ is still undecided and the remaining literals $C \setminus l_i$ evaluate to $\bot$, the rule forces the DPLL solver to explore only the case in which l_i is assigned to $\top$, as C would otherwise remain unsatisfied.

Conflict-Driven-Clause-Learning (CDCL) Algorithm [MSLM09]. The CDCL algorithm is an evolution of DPLL that replaces the original *chronological backtracking* mechanism with a *backjumping* mechanism based on *conflict analysis* and *clause learning*.

Similarly to DPLL, the procedure starts with an empty assignment μ and it attempts to extend μ to a complete assignment μ' that satisfies the input formula φ. In CDCL, the search proceeds in a stack-based loop and it is organised in decision levels, starting from level 0. The latter contains the original set of input clauses φ after the application of pre-processing and unit-propagation. For each new level, the CDCL engine performs three steps: Decision, Boolean Constraint Propagation (BCP) and Backjumping and Learning.

In a Decision step, the (partial) assignment μ is extended with a new decision literal l, that is heuristically selected among the set of literals in φ that are still undecided.

Then, the CDCL algorithm applies BCP to iteratively deduce, by means of unit-propagation, all literals l' that are implied by μ. Each of these literals is tagged with the clause that caused

its propagation, called *antecedent clause*, and added to μ. BCP ends either when there are no more literals that can be deduced or when μ causes some clause $C \in \varphi$ to be falsified. In the first case, the search continues with a new decision if there is still some undecided literal in φ, and terminates with SAT otherwise. If instead the search incurs in a conflict, then a step of Backjumping and learning is performed.

To perform a Backjump, the implication graph —which is implicitly defined on the search-stack by the set of *antecedent clauses*— has to be traversed to identify the subset η of literals that caused C to be falsified (conflict set). Then, the conflict clause $C' \stackrel{\text{def}}{=} \neg\eta$ is learned, and the search backjumps to some previous decision level $blevel$ in which the procedure would have done something different if the new clause C' was already known (for details on some strategies for picking $blevel$ see [MsS99, ZMMM01]). Compared with the chronological backtracking approach of DPLL, backjumping is a lot more effective because it immediately jumps to the place in which a mistake was performed, thus avoiding a lot of useless search.

Learning C' guarantees that the same mistake will not be repeated again in the future, because as soon as all-but-one literals in η are decided, then the remaining literal is unit-propagated to $\bot$. However, clause learning can also heavily impact the overall performance due to generating a large number of new clauses. For this reason, CDCL solvers typically employ a technique called *clause discharging*, that heuristically drops unnecessary learned clauses from the formula stack, e.g. using their size and activity as relevance indicators.

Finally, the search ends with UNSAT when two contradictory literals l and $\neg l$ are assigned at level 0 as a result of unit-propagation.

2.2 Satisfiability Modulo Theories

In the following, we provide a brief overview to *Satisfiability Modulo Theories* (SMT), focusing for the most part on lazy SMT. The material presented in this section is standard in SMT, and it is mostly taken from [NOT06, Seb07, BSST09]. Hence, we refer the reader to these publications for a broader, and drastically more rigorous, introduction to the topic.

Basics. We assume that the reader is familiar with standard first order logic, and otherwise refer to any book that gives an introduction on the topic (e.g. [vD94]).

In the following, let Σ be a first-order signature containing function and predicate symbols with their arities, and $\mathcal{V}$ be a set of variables. A 0-ary function symbol c is called a *constant*. A 0-ary predicate symbol A is called a *Boolean atom*. A Σ-*term* is either a variable in $\mathcal{V}$ or it is built by applying function symbols in Σ to Σ-terms. If $t_1, ..., t_n$ are Σ-terms and P is a predicate symbol, then $P(t_1, ..., t_n)$ is a Σ-*atom*. If l and r are two Σ-terms, then the Σ-atom $l = r$ is called a Σ-*equality* and $\neg(l = r)$ (also written as $l \neq r$) is called a Σ-*disequality*. A

Σ-*formula* φ is built in the usual way out of the universal and existential quantifiers $\forall$, $\exists$, the Boolean connectives $\wedge$, $\neg$, and Σ-atoms. We use the standard Boolean abbreviations: "$\varphi_1 \vee \varphi_2$" for "$\neg(\neg\varphi_1 \wedge \neg\varphi_2)$", "$\varphi_1 \rightarrow \varphi_2$" for "$\neg(\varphi_1 \wedge \neg\varphi_2)$", "$\varphi_1 \leftarrow \varphi_2$" for "$\neg(\neg\varphi_1 \wedge \varphi_2)$", "$\varphi_1 \leftrightarrow \varphi_2$" for "$\neg(\varphi_1 \wedge \neg\varphi_2) \wedge \neg(\neg\varphi_1 \wedge \varphi_2)$", "$\top$" [resp. "$\bot$"] for the true [resp. false] constant. A Σ-*literal* is either a Σ-atom (a *positive literal*) or its negation (a *negative literal*). The set of Σ-atoms and Σ-literals occurring in φ are denoted by $Atoms(\varphi)$ and $Lits(\varphi)$ respectively. A formula φ is said to be *quantifier-free* if it does not contain quantifiers, and *ground* if it has no free variables. A disjunction of literals is called a *clause*.

Notationally, we use the greek letters φ, ψ to represent Σ-formulas, the capital letters A_i's and B_i's to represent Boolean atoms, and the Greek letters α, β, γ to represent Σ-atoms in general, the letters l_i's to represent Σ-literals. If l is a negative Σ-literal $\neg\beta$, then by "$\neg l$" we conventionally mean β rather than $\neg\neg\beta$.

We also assume that the reader is familiar with the usual first-order notions of interpretation, satisfiability, validity, logical consequence and theory, as given, e.g., in [vD94]. We write $\Gamma \models \varphi$ to denote that φ is a logical consequence of the (possibly infinite) set Γ of formulas.

Throughout this dissertation, for simplicity and if not specified otherwise, we shall refer —with a small abuse of notation— to predicates of arity zero as *Boolean variables*, and to uninterpreted constants as *theory variables* (or simply as variables, when the meaning is clear from the context).

Theory ($\mathcal{T}$). A Σ*−theory* is a set of first-order sentences with signature Σ. We use the symbol $\mathcal{T}$ to denote a Σ−-theory. We use the prefix "$\mathcal{T}$-" to denote any atom/clause/formula that is defined solely in terms of elements in the theory $\mathcal{T}$. In this dissertation, we consider only *quantifier-free* first-order theories with equality, meaning that the symbol "$=$" is interpreted as the identity relation in each $\mathcal{T}$[1].

A Σ-structure $\mathcal{I}$ is a model of a Σ-theory $\mathcal{T}$ if $\mathcal{I}$ satisfies every sentence in $\mathcal{T}$. A Σ-formula is *satisfiable* in $\mathcal{T}$ (or $\mathcal{T}$*-satisfiable*) if it is satisfiable in a model of $\mathcal{T}$. We write $\gamma \models_{\mathcal{T}} \varphi$ to denote $\mathcal{T} \cup \gamma \models \varphi$. Two Σ-formulas φ and ψ are $\mathcal{T}$*-equisatisfiable* if and only if φ is $\mathcal{T}$-satisfiable if and only if ψ is $\mathcal{T}$-satisfiable. We call *Satisfiability Modulo (the) Theory* $\mathcal{T}$, *SMT* ($\mathcal{T}$), the problem of establishing the $\mathcal{T}$-satisfiability of Σ-formulas, for some background theory $\mathcal{T}$. We call a *theory solver* for $\mathcal{T}$ ($\mathcal{T}$*-solver*) any procedure that can establish whether any given finite conjunction of quantifier-free Σ-literals is $\mathcal{T}$-satisfiable or not.

Henceforth, for simplicity and if not specified otherwise, we may omit the "Σ-" prefix from term, formula, theory, models, etc. Moreover, by "formulas", "atoms" and "literals" we implic-

[1]Therefore, as a consequence of focusing only on the *quantifier-free* fragment of each theory $\mathcal{T}$ and in absence of any ambiguity, throughout this document we drop —with a small abuse of notation— the "Quantifier Free" (i.e. "QF") prefix from the name of each SMT theory being considered (e.g. QF_LIRA becomes $\mathcal{LIRA}$).

itly refer to *quantifier-free* formulas, atoms and literals respectively.

Given a disjunction $\bigvee_{i=1}^{n} x_i = y_i$, where x_i, y_i are variables, a conjunction Γ of $\mathcal{T}$-literals in a theory $\mathcal{T}$ is said to be *convex* if and only if it is always true that $\Gamma \models_{\mathcal{T}} \bigvee_{i=1}^{n} x_i = y_i$ if and only if $\Gamma \models_{\mathcal{T}} x_i = y_i$ for some $i \in [1, n]$. A theory $\mathcal{T}$ is said to be *convex* if all possible conjunctions of its $\mathcal{T}$-literals are convex in $\mathcal{T}$. A theory $\mathcal{T}$ is said to be *stably-infinite* if and only if for each $\mathcal{T}$-satisfiable formula φ there exists a model of $\mathcal{T}$ whose domain is infinite and that satisfies φ. Any convex theory $\mathcal{T}$ whose models' domains have all cardinality strictly greater than one is stably-infinite [BDS02].

Abstraction/Refinement ($\mathcal{T}2\mathcal{B}$/$\mathcal{B}2\mathcal{T}$). Given a first-order $\mathcal{T}$-formula φ, its *propositional/Boolean abstraction* φ^p is obtained by replacing each $\mathcal{T}$-atom in φ with a fresh Boolean constant. The $\mathcal{T}$-formula φ is called *refinement* of φ^p. We assume the availability of a mapping $\mathcal{T}2\mathcal{B}$ ("theory to Boolean") from theory atoms to fresh Boolean constants and its inverse $\mathcal{B}2\mathcal{T}$ ("Boolean to theory") to get the propositional abstraction φ^p from φ and vice versa.

Truth Assignment. We call a *truth assignment* μ for a $\mathcal{T}$-formula φ a truth value assignment to the $\mathcal{T}$-atoms of φ. A truth assignment μ is said to be *total* if it assigns a value to every atom in φ, and *partial* otherwise. Syntactically identical instances of the same $\mathcal{T}$-atom are always assigned identical truth values; syntactically different $\mathcal{T}$-atoms, e.g., $(t_1 \geq t_2)$ and $(t_2 \leq t_1)$, are treated differently and may thus be assigned different truth values.

We represent a truth assignment μ for φ as a set of $\mathcal{T}$-literals

$$\{\alpha_1, ..., \alpha_N, \neg\alpha_{N+1}, ..., \neg\alpha_M, A_1, .., A_R, \neg A_{R+1}, ..., \neg A_S\}$$

where the α_i's are Σ-atoms and A_i's are Boolean propositions. Positive literals α_i, A_j mean that the corresponding atoms is assigned to true, negative literals $\neg\alpha_i$, $\neg A_j$ mean that the corresponding atom is assigned to false. If $\mu_2 \subseteq \mu_1$, then we say that μ_1 *extends* μ_2 and that μ_2 *subsumes* μ_1. Sometimes we represent a truth assignment also as the formula given by the conjunction of its literals. Notationally, we use the Greek letters μ, η to represent truth assignments.

Satisfiability. Given a truth assignment μ, we denote with μ^p the *Boolean abstraction* of μ, that is, $\mu^p \stackrel{\text{def}}{=} \mathcal{T}2\mathcal{B}(\mu)$. Given a total truth assignment μ for φ, we say that μ *propositionally satisfies* φ, written $\mu \models_p \varphi$, if and only if $\mu^p \models \varphi^p$, where φ^p is the Boolean abstraction of φ. We say that a partial truth assignment μ *propositionally satisfies* φ if and only if all the total truth assignments for φ that extend μ propositionally satisfy φ. We say that φ is *propositionally satisfiable* if and only if there exist an assignment $\mu \models_p \varphi$. We say that φ is *propositionally unsatisfiable* if no such μ exists. We say that φ is *$\mathcal{T}$-satisfiable* if and only if there exists some total truth assignment μ for φ such that μ propositionally satisfies φ and μ is $\mathcal{T}$-satisfiable. We say that φ is *$\mathcal{T}$-unsatisfiable* if there exist no such $\mathcal{T}$-satisfiable truth assignment μ for φ that

propositionally satisfies φ. A structure $\mathcal{M}$ is said to be a model of a theory $\mathcal{T}$ if $\mathcal{M}$ satisfies every formula in $\mathcal{T}$. We say that a formula φ is *$\mathcal{T}$-satisfiable* if it is satisfiable in a model of $\mathcal{T}$.

Definition 2.2.1. *($\mathcal{T}$-Solver [BSST09]). Given a first-order theory $\mathcal{T}$ for which the (ground) satisfiability problem is decidable and a conjunction/set of* theory literals *($\mathcal{T}$-literals) μ, a* theory solver for $\mathcal{T}$ *($\mathcal{T}$-Solver) is any decision procedure that is capable to decide the satisfiability of μ in $\mathcal{T}$.*

In the case μ is $\mathcal{T}$-unsatisfiable, a typical $\mathcal{T}$-Solver not only returns UNSAT, but also a *theory conflict set* η such that $\eta \subseteq \mu$ and η is $\mathcal{T}$-unsatisfiable. We note that a conflict set η does not need to be minimal and that its negation $\neg\eta$ is called *theory conflict clause*. If instead μ is $\mathcal{T}$-satisfiable then the $\mathcal{T}$-Solver returns SAT. In addition, it may also be able to generate one (or more) deductions of the form $\{l_1, ..., l_n\} \models_{\mathcal{T}} l$ such that $\{l_1, ..., l_n\} \subseteq \mu$ and l is an unassigned $\mathcal{T}$-literal (i.e. $l \notin \mu$). The formula $(\bigvee_{i=1}^{n} \neg l_i \vee l)$ is called a *theory-deduction clause*. Both theory-conflict clauses and theory-deduction clauses are valid in $\mathcal{T}$, and are therefore called *theory lemmas* or *$\mathcal{T}$-lemmas*.

Definition 2.2.2. *(Satisfiability Modulo Theories (SMT) [BSST09]). Let $\mathcal{T} \stackrel{def}{=} \bigcup_i \mathcal{T}_i$, where each pair of theories $\mathcal{T}_i, \mathcal{T}_j$ in $\mathcal{T}$ is signature-disjoint, meaning that $\mathcal{T}_i$ and $\mathcal{T}_j$ share no symbol except for the equality symbol "=". Then, Satisfiability Modulo Theories is the problem of deciding the satisfiability of* Boolean combinations *of propositional atoms and theory atoms that belong to $\mathcal{T}$.*

In §2.2.1 we provide a short introduction to the salient aspects of the so-called *"lazy"* approach [Seb07, BSST09], also known as "DPLL ($\mathcal{T}$)" [NOT06], that is implemented by modern SMT solvers. The *incremental* extension of *"lazy"* SMT solving is explored in §2.2.2, followed by §2.2.3 with a lightweight excursus of some notable SMT theories $\mathcal{T}$ that are of interest for this dissertation, introducing concepts that will be used later on. We conclude with *combination of theories* in SMT in §2.2.4.

2.2.1 The "lazy" SMT Scheme

A "lazy" SMT solver combines combines a propositional SAT solver based on the DPLL algorithm[2] with a number of $\mathcal{T}$-Solvers, (at least) one for each theory $\mathcal{T}$ of interest. With this

[2]Modern DPLL solvers often implement the CDCL techniques described in §2.1. For this reason, in this thesis we often use the name CDCL to refer to the SAT engine embedded in a "lazy" SMT solver.

```
function T-DPLL (T-formula φ, T-assignment &μ)
 1: res := T-PREPROCESS(φ, μ)
 2: if (res == conflict) then
 3:     return UNSAT
 4: φ^p := T2B(φ)
 5: μ^p := T2B(μ)
 6: while true do
 7:     T-DECIDE_NEXT_BRANCH(φ^p, μ^p)
 8:     while true do
 9:         res := T-DEDUCE(φ^p, μ^p)
10:         if (res == SAT) then
11:             μ := B2T(μ^p)
12:             return SAT
13:         else if (res == conflict) then
14:             ⟨blevel, η⟩ := T-ANALYZE_CONFLICT(φ^p, μ^p)
15:             if (blevel < 0) then
16:                 return UNSAT
17:             else
18:                 T-BACKTRACK(blevel, φ^p, μ^p, η^p)
19:         else
20:             break
```

Figure 2.1: An online schema of $\mathcal{T}$-DPLL based on modern DPLL [Seb07].

approach, the DPLL engine is used to enumerate truth assignments μ_i^p that propositionally satisfy the Boolean abstraction φ^p of the input formula φ (i.e. $\mu_i^p \models_p \varphi^p$), whereas the $\mathcal{T}$-Solvers are used to check that μ_i, where $\mu_i \stackrel{\text{def}}{=} \mathcal{B}2\mathcal{T}(\mu_i^p)$, is $\mathcal{T}$-satisfiable.

$\mathcal{T}$-DPLL Algorithm. The basic schema of a "lazy" SMT solver is shown in algorithm 2.1. The procedure takes as inputs a $\mathcal{T}$-formula φ and an (initially empty) set of $\mathcal{T}$-literals μ, that is updated during the search.

The search starts by invoking $\mathcal{T}$-PREPROCESS(), a procedure that transforms φ into a simpler and equi-satisfiable formula —possibly in CNF— and correspondingly updates μ if necessary (line 1). In the case the SMT solver detects that φ is unsatisfiable while performing this simplification step, the search is early-terminated with UNSAT (lines 2-3). Otherwise, the SMT solver uses the function $\mathcal{T}2\mathcal{B}$ to generate the Boolean abstractions φ^p and μ^p starting from φ and μ (lines 4-5), so that it can apply the SAT based techniques described in §2.1.

After entering the main loop (lines 6-20), the first step is to call $\mathcal{T}$-DECIDE_NEXT_BRANCH() to extend μ^p with some currently unassigned literal l from φ^p. Here, the literal l is called *decision literal* and it is picked according to some heuristic function that might take into considera-

tion the semantics of $\mathcal{T}$. The number of decision literals in μ after performing this step is called *decision level* of l.

Inside the inner loop (lines 8-20), the function $\mathcal{T}$-DEDUCE() is used to iteratively deduce all literals l' that derive from the current truth assignment (i.e. $\varphi \wedge \mu \models_p l'$) at line 9. This step is akin to BCP in §2.1 and it keeps running up until when one of the following circumstances arises:

(i) μ^p satisfies φ^p (i.e. $\mu^p \models_p \varphi^p$), so that $\mathcal{T}$-DEDUCE() checks the $\mathcal{T}$-satisfiability of $\mathcal{B}2\mathcal{T}(\mu^p)$ by invoking the appropriate $\mathcal{T}$-*solver*(s). The result of $\mathcal{T}$-DEDUCE() is SAT if every $\mathcal{T}$-*solver* returns SAT, and conflict otherwise.

(ii) μ^p propositionally violates φ^p (i.e. $\mu^p \wedge \varphi^p \models_p \bot$), so that $\mathcal{T}$-DEDUCE() returns UNSAT.

(iii) no more literals can be deduced, so that $\mathcal{T}$-DEDUCE() returns UNKNOWN.

The return value of $\mathcal{T}$-DEDUCE(), res, is then subsequently handled as follows:

(i) if it is equal to SAT, then $\mathcal{T}$-DPLL terminates with the same value after μ^p is refined into a set of $\mathcal{T}$-literals μ with the aid of function $\mathcal{B}2\mathcal{T}()$ (lines 10-12).

(ii) if it is equal to conflict, then the SMT solver encountered a conflict either at the propositional or at the theory level. In both cases, the search proceeds similarly to the CDCL approach described in §2.1. It invokes a procedure named $\mathcal{T}$-ANALYZE_CONFLICT() to identify a subset η^p of μ^p —a.k.a. the *conflict set*— that is causing the conflict and the decision level blevel to which the search has to backtrack (line 14). If the conflict is not a logical consequence of a previous branching step, i.e. $blevel$ is equal to 0, then the search terminates with UNSAT (lines 15-16). Otherwise, $\mathcal{T}$-BACKTRACK() extends φ with the *conflict clause* $\neg\eta$ and backjumps to $blevel$. Both φ and μ are updated accordingly (lines 17-18).

(iii) if it is equal to UNKNOWN, then the SMT solver exists the inner loop and proceeds with a new *Decision* by jumping at the next iteration of the outer loop (lines 19-20).

The search continues up until when a $\mathcal{T}$-consistent set of literals μ is generated, meaning that φ is $\mathcal{T}$-satisfiable, or when every possible assignment leads to a conflict, meaning that φ is $\mathcal{T}$-unsatisfiable.

Enhancements. In the following, we provide a brief list of the most important enhancements that are typically adopted to improve the performance of the basic online $\mathcal{T}$-DPLL schema shown in algorithm 2.1. We refer the reader to [NOT06, Seb07, BSST09] for more details and techniques.

- *Early Pruning* (EP). *$\mathcal{T}$-solvers* are invoked on $\mathcal{B}2\mathcal{T}$ (μ^p) even when μ^p is still a partial truth assignment, possibly (at least) once before each new *Decision*, so that in the case of a conflict the search can immediately backtrack without exploring any of the (many) possible extensions of μ^p. On the one hand, this enhancement allows the SMT solver to discover $\mathcal{T}$-conflicts —that are typically small— earlier in the search, thus sparing lots of search. On the other hand, invoking any *$\mathcal{T}$-solver* is typically more expensive than simple BCP. To lower the impact of this approach, *$\mathcal{T}$-solvers* are typically designed to be *incremental* and "remember" their computation status from one call to the other, so that they do not have to start from scratch each time μ is extended by a new literal l.
- *Weak Early Pruning* (WEP) [Seb07]. Allows *$\mathcal{T}$-solvers* to perform only an *approximate*, but cheaper, satisfiability check during EP calls, thus reducing the overhead of EP as a whole. In practice, during EP calls *$\mathcal{T}$-solvers* are allowed to return SAT even when the current truth assignment μ is $\mathcal{T}$-inconsistent, as long as they are able to identify such inconsistency during non-EP calls.
- *$\mathcal{T}$-propagation* [NOT06]. Given $\eta \stackrel{\text{def}}{=} \{l_1, ..., l_n\}$ such that $\eta \subseteq \mu$ and an unassigned literal l^p corresponding to an atom in φ, if the *$\mathcal{T}$-solver* is able to deduce that $\eta \models_{\mathcal{T}} l$, where $l \stackrel{\text{def}}{=} \mathcal{B}2\mathcal{T}(l^p)$, then l^p is unit-propagated by extending μ^p with the implied literal. In addition, the *$\mathcal{T}$-deduction clause* $(\bigvee_{i=1}^{n} \neg l_i \vee l)$ can be permanently learned by the SMT solver to be used in backjumping and learning. When relatively cheap, this technique can have cascade benefits to the overall performance, as $\mathcal{T}$-propagating one literal l may allow new literals to be assigned by BCP or deduced by a new round of $\mathcal{T}$-propagation.
- *Layering* [BBC$^+$05c, BCF$^+$07]. Each *$\mathcal{T}$-solver* may be organised in a *layered hierarchy* $S_1, ..., S_N$ of increasing expressibility and complexity, so that each S_i is able to decide a theory $\mathcal{T}_i$ that is a sub-theory of $\mathcal{T}_{i+1}$. Greater expressibility and complexity entails more expensive procedures for deciding the satisfiability of μ over the fragment of $\mathcal{T}$ being supported. In this architecture, only the top-most solver S_N is able to decide the full theory $\mathcal{T}$. Layering plays a significant role when μ is $\mathcal{T}$-inconsistent, because the *$\mathcal{T}$-solver* can return UNSAT as soon as some solver S_i reveals such inconsistency, without a need to invoke any of the more expensive engines.
- *Splitting on-demand* [BNOT06]. A *$\mathcal{T}$-solver* is allowed to return UNKNOWN plus a set of *new $\mathcal{T}$-lemmas containing new $\mathcal{T}$-atoms*, whose abstraction is then handled by the SAT

engine at the Boolean level, as with the other clauses in φ. This enhancement leverages the very efficient techniques implemented within the SAT engine (e.g. conflict-driven backjumping and learning) to perform *disjunctive reasoning* that would otherwise have to be directly handled by the $\mathcal{T}$*-solver* whenever this is necessary. This technique does not only positively impact on the performance, but it also allows for simpler $\mathcal{T}$*-solver* implementations.

- *Pure Literal Filtering* [Seb07]. Given any $\mathcal{T}$-atom that occurs only positively [resp. negatively] in the input formula φ, the SMT solver is allowed to (safely) drop every negative [resp. positive] occurrence of it from μ before handing it over to the $\mathcal{T}$*-solver*. Intuitively, none of these $\mathcal{T}$-atoms is crucial to determining the $\mathcal{T}$-satisfiability of μ. As a result, the $\mathcal{T}$*-solver* benefits both from being asked to handle fewer literals at a time, and also from the fact that removing "useless" $\mathcal{T}$-literals from μ increases the chances that the tracked set of $\mathcal{T}$-literals is found to be consistent.

2.2.2 Incremental SMT

A common feature that characterizes modern SMT solvers is the availability of a *stack-based incremental interface* (see e.g. [ES04]), that allows for pushing/popping subformulas φ_i into an internal stack of formulas $\Phi \stackrel{\text{def}}{=} \{\phi_1, ..., \phi_k\}$ and then to incrementally check the satisfiability of $\varphi \stackrel{\text{def}}{=} \bigwedge_{i=1}^{k} \phi_i$.

Efficient incremental SMT is possible thanks to a *status-based* design that applies to both the $\mathcal{T}$-DPLL engine as well as to any $\mathcal{T}$*-solver*. This design preserves the search status from one incremental call to the other, such as learned clauses and the phase-saving value of each Boolean literal. As a result, when invoked on φ, the SMT solver can reuse a clause C that was learned during a previous call on some φ', provided that (1) C was derived only from clauses that are still in φ and that (2) C was not discharged in the meantime. In the particular case in which $\varphi' \subseteq \varphi$, then the SMT solver is able to reuse all clauses that were learned while checking the satisfiability of φ'. An additional benefit of the *status-based* design is that it allows for an efficient restore of the previous state upon a subformulas discharge event.

A possible approach for incremental SMT (used, e.g., by MATHSAT5 [CGSS13b]), follows. First, the stack of formulas Φ, which we defined as $\{\phi_1, ..., \phi_k\}$, is rewritten into the new stack $\Phi' \stackrel{\text{def}}{=} \{A_1 \rightarrow \phi_1, ..., A_k \rightarrow \phi_k\}$, where each A_i is a fresh Boolean variable. Then, the SMT solver checks the satisfiability of Φ' under the assumption of the variables $\{A_1, ..., A_k\}$, so that every learned clause C that is derived from some ϕ_i is in the form $\neg A_i \vee C'$ [ES04]. When a subformula ϕ_i is popped from the stack of formulas, the corresponding Boolean variable A_i is no longer assumed in subsequent satisfiability checks. As a result of dropping A_i from the set

of assumptions, any learned clause C of the form $\neg A_i \vee C'$ becomes inactive, meaning that it no longer contributes to the satisfiability of the input formula and thus it can be ignored by the SMT solver. Hence, a clause can be safely stored in memory from one call to the other, at least up until when, after becoming inactive, it is automatically garbage-collected to free up some space.

2.2.3 Theories of Interest

In the following we take a closer look to a short list of notable theories $\mathcal{T}$ that are typically handled by SMT solvers and are also of interest for Optimization Modulo Theories in the context of this dissertation. We refer the interested reader to [Seb07, BSST09] and to the SMT-LIB website [smt] for more theories $\mathcal{T}$ and a more formal (and detailed) presentation.

Equality and Uninterpreted Functions ($\mathcal{EUF}$).

This is a first order theory for quantifier-free formulas that only deals with the following equality (2.1a) and congruence (2.1b) axioms, defined for every function symbol f and predicate symbol P:

$$\forall x.(x = x), \forall x, y.(x = y \rightarrow y = x), \forall x, y, z.((x = y \wedge y = z) \rightarrow x = z) \tag{2.1a}$$

$$\begin{aligned} &\forall x_1, ..., x_n, y1, ..., y_n.((\bigwedge\nolimits_{i=1}^{n} x_i = y_i) \rightarrow f(x_1, ..., x_n) = f(y_1, ..., y_n)) \\ &\forall x_1, ..., x_n, y1, ..., y_n.((\bigwedge\nolimits_{i=1}^{n} x_i = y_i) \rightarrow P(x_1, ..., x_n) \leftrightarrow P(y_1, ..., y_n)) \end{aligned} \tag{2.1b}$$

The $\mathcal{EUF}$ theory is both stably-infinite and convex. Moreover, given a quantifier-free set of literals, $\mathcal{EUF}$-satisfiability is both decidable and polynomial.

Linear Arithmetic ($\mathcal{LIRA}$).

The theory of *Linear Arithmetic* ($\mathcal{LIRA}$) is the quantifier-free first order theory with equality whose atoms are in the form $(a_1 \cdot x_1 + ... + a_n \cdot x_n \bowtie a_0)$, such that $\bowtie \in \{\leq, <, \neq, =, \geq, >\}$, each a_i is an (interpreted) constant symbol that belongs to either the Rational or Integer domain and each x_i is either a Rational or an Integer variable. The restriction of $\mathcal{LIRA}$ to the Rational and Integer domain only is the theory of *Linear Rational Arithmetic* ($\mathcal{LRA}$) and *Linear Integer Arithmetic* ($\mathcal{LIA}$) respectively.

Linear Rational Arithmetic ($\mathcal{LRA}$). The theory of $\mathcal{LRA}$ is both stably-infinite and convex. Moreover, the $\mathcal{LRA}$-satisfiability of quantifier-free sets of $\mathcal{LRA}$-atoms is both decidable and

polynomial [Kha79]. Modern SMT solver often implement some type of Simplex-based procedure for $\mathcal{LRA}$-satisfiability [DdM06b], that has the benefits of being efficient, incremental and backtrackable, it allows for aggressive $\mathcal{T}$-deductions and it directly handles *strict inequalities* [DdM06b].

Linear Integer Arithmetic ($\mathcal{LIA}$). The theory of $\mathcal{LIA}$ is stably-infinite and non-convex. In contrast with $\mathcal{LRA}$, the $\mathcal{LIA}$-satisfiability of quantifier-free sets of $\mathcal{LIA}$-atoms is decidable and NP-complete [Pap81]. Several algorithms have been proposed in the past, often combined with one another, to handle $\mathcal{LIA}$-satisfiability efficiently: Simplex-based search with Branch&Bound [Sch99], Gomory's cutting planes method [DM06a], the Omega test [Pug92] based on the Fourier-Motzkin algorithm, and others (e.g. [DDA09, Gri12]).

Remark 2.2.1. In the context of SMT solving, which has its main applications in the domains of formal verification and model checking, it is of absolute importance that the SMT solver is able to guarantee the correctness of a result. For this reason, the *Linear Arithmetic* algorithms employed by SMT solvers are typically built on top of *infinite-precision-arithmetic* software packages, thus avoiding incorrect results due to numerical errors and to overflows.

Bit-Vectors ($\mathcal{BV}$).

The theory of fixed-width *Bit-Vectors* ($\mathcal{BV}$) is a quantifier-free first order theory with equality that is used, for instance, to represent *Register Transfer Level* (RTL) hardware circuits at a higher, modular, level than what is possible with a purely propositional approach (e.g. "bit blasting"). In addition, the $\mathcal{BV}$ theory can also be used to deal with problems steaming from the software verification domain (e.g. [GD07]).

In the $\mathcal{BV}$ theory, a *bit* is a Boolean variable that can be interpreted as 0 or 1 and a $\mathcal{BV}$ variable $\mathbf{v}^{[n]}$ of width n is a sequence of n bits $[\mathsf{obj}[0], ..., \mathsf{obj}[n-1]]$, where $v[0]$ is the *Most Significant Bit* (MSB) and $v[n-1]$ is the *Least Significant Bit* (LSB)[3] A $\mathcal{BV}$ constant of width n is an interpreted vector of n values in $\{0, 1\}$. We $\overline{overline}$ a bit value or a $\mathcal{BV}$ value to denote its complement (e.g., $\overline{[11010010]}$ is $[00101101]$). A $\mathcal{BV}$ variable/constant of width n can be *unsigned*, in which case its domain is $[0, 2^n - 1]$, or *signed*, that we assume to comply with the *two's complement* representation, so that its domain is $[-2^{(n-1)}, 2^{(n-1)} - 1]$. Therefore, the vector $[11111111]$ can be interpreted either as the unsigned $\mathcal{BV}$ constant $\mathbf{255}^{[8]}$ or as the signed $\mathcal{BV}$ constant $-\mathbf{1}^{[8]}$. Following the SMT-LIBv2 standard [smt], we may also represent a $\mathcal{BV}$

[3] In the literature, $v[0]$ and $v[n-1]$ commonly represent the LSB and the MSB respectively. We use the opposite notation because we always reason from the MSB down to the LSB.

constant in *binary* (e.g. $\mathbf{28}^{[8]}$ is written $\#b00011100$) or in *hexadecimal* (e.g. $\mathbf{28}^{[8]}$ is written $\#x1C$) form.

A $\mathcal{BV}$ term is built from $\mathcal{BV}$ constants, variables and interpreted $\mathcal{BV}$ functions that represent standard RTL operators: word concatenation (e.g. $\mathbf{3}^{[8]} \circ \mathbf{x}^{[8]}$), subword selection (e.g. $(\mathbf{3}^{[8]}[6:3])^{[4]}$), modulo-n sum and multiplication (e.g. $\mathbf{x}^{[8]} +_8 \mathbf{y}^{[8]}$ and $\mathbf{x}^{[8]} \cdot_8 \mathbf{y}^{[8]}$), bit-wise operators (like, e.g., $\mathbf{and}_n$, $\mathbf{or}_n$, $\mathbf{xor}_n$, $\mathbf{not}_n$), left and right shift $<<_n$, $>>_n$. A $\mathcal{BV}$ atom can be built by combining $\mathcal{BV}$ terms with interpreted predicates like $\geq_n$, $<_n$ (e.g. $\mathbf{0}^{[8]} \geq_8 \mathbf{x}^{[8]}$) and equality. We refer the reader to [smt, Had15] for further details on the syntax and semantics of the Bit-Vector theory.

The theory of $\mathcal{BV}$ is non-stably infinite and non-convex. Furthermore, the $\mathcal{BV}$-satisfiability of sets of quantifier-free $\mathcal{BV}$-atoms is decidable and NP-complete. In the context of SMT solving, two main families of approaches have been proposed. In the *eager* approach, $\mathcal{BV}$ terms and constraints are encoded into SAT via bit-blasting [GD07, BB09, Bru09, Had15, NPFB15, Nie17]. Instead, in the *lazy* approach, the $\mathcal{BV}$ encoding is not immediately expanded —to avoid any scalability issue— and the $\mathcal{BV}$ solver is composed by a layered set of techniques, each of which deals with a sub-portion of the $\mathcal{BV}$ theory [BD02, BBC+05a, BCF+07, Had15]. The empirical evidence of [HBJ+14] has shown that the two approaches are complementary to one another and that $\mathcal{BV}$ solving can benefit from a *portfolio* solution combining both techniques in one solver.

Floating-Point ($\mathcal{FP}$).

The theory of *Floating-Point Numbers* ($\mathcal{FP}$), [smt, RW10, BTRW15], is a quantifier-free first order theory with equality that is based on the IEEE standard 754-2008, [iee08], for floating-point arithmetic, restricted to the binary case. A major difference with [iee08] is that the theory of $\mathcal{FP}$ defined in [RW10, BTRW15] allows for every possible exponent and significand length.

A $\mathcal{FP}$ sort is an indexed nullary sort identifier of the form `(_ FP` $\langle ebits \rangle$ $\langle sbits \rangle$`)` such that both $ebits$ and $sbits$ are positive integers greater than one, $ebits$ defines the number of bits in the exponent and $sbits$ defines the number of bits in the significand, including the hidden bit.

A $\mathcal{FP}$ variable $\mathbf{v}$ with sort `(_ FP` $\langle ebits \rangle$ $\langle sbits \rangle$`)` can be indifferently viewed as a vector of $n \stackrel{\text{def}}{=} ebits + sbits$ bits, where $v[0]$ is the *Most Significant Bit* (MSB) and $v[n-1]$ is the *Least Significant Bit* (LSB), or as a triplet of Bit-Vectors $\langle \mathbf{sign}, \mathbf{exp}, \mathbf{sig} \rangle$ such that $\mathbf{sign}$ is a $\mathcal{BV}$ of size 1, $\mathbf{exp}$ is a $\mathcal{BV}$ of size $ebits$ and $\mathbf{sig}$ is a $\mathcal{BV}$ of size $sbits - 1$. A $\mathcal{FP}$ constant is a triplet of $\mathcal{BV}$ constants. Given a fixed floating-point sort, i.e. a pair $\langle ebits, sbits \rangle$, the following $\mathcal{FP}$ constants are implicitly defined:

value	**Symbol**	**$\mathcal{BV}$ Repr.**
plus infinity	`(_ +oo` *<ebits>* *<sbits>*`)`	`(fp #b0 #b1...1 #b0...0)`
minus infinity	`(_ -oo` *<ebits>* *<sbits>*`)`	`(fp #b1 #b1...1 #b0...0)`
plus zero	`(_ +zero` *<ebits>* *<sbits>*`)`	`(fp #b0 #b0...0 #b0...0)`
minus zero	`(_ -zero` *<ebits>* *<sbits>*`)`	`(fp #b1 #b0...0 #b0...0)`
not-a-number	`(_ NaN` *<ebits>* *<sbits>*`)`	`(fp t #b1...1 s)`

where `t` is either 0 or 1 and `s` is a $\mathcal{BV}$ that contains at least a 1.

Setting aside special $\mathcal{FP}$ constants, the remaining $\mathcal{FP}$ values can be classified to be either normal or subnormal (a.k.a. denormal) [iee08]. A $\mathcal{FP}$ number is said to be *subnormal* when every bit in its exponent is equal to zero, and *normal* otherwise. The significand of a normal $\mathcal{FP}$ number is always interpreted as if the leading binary digit is equal 1, whereas for denormalized $\mathcal{FP}$ values the leading binary digit is always 0. This allows for the representation of numbers that are closer to zero, although with reduced precision.

Example 2.2.1. *Let x be the normal $\mathcal{FP}$ constant `(_ FP #b0 #b1100 #b0101000)`, and y be the subnormal $\mathcal{FP}$ constant `(_ FP #b0 #b0000 #b0101000)`, so that their corresponding sort is `(_ FP <4> <8>)`. Then, according to the semantics defined in the IEEE standard 754-2008 [iee08], the floating-point value of x in decimal notation is given by:*

$$\begin{aligned} x &= (-1)^0 \cdot 2^{(12-7)} \cdot \left(1 + \sum_{i=1}^{7} \left(x[4+i] \cdot 2^{-i}\right)\right) \\ &= 1 \cdot 2^5 \cdot \left(1 + \frac{1}{2^2} + \frac{1}{2^4}\right) \\ &= 2^5 \cdot \frac{2^4 + 2^2 + 1}{2^4} \\ &= 2 \cdot 21 \\ &= 42 \end{aligned}$$

and the value of y is given by:

$$\begin{aligned} y &= (-1)^0 \cdot 2^{(0-7+1)} \cdot \left(0 + \sum_{i=1}^{7} \left(y[4+i] \cdot 2^{-i}\right)\right) \\ &= 1 \cdot 2^{-6} \cdot \left(\frac{1}{2^2} + \frac{1}{2^4}\right) \\ &= \frac{1}{2^6} \cdot \frac{2^2 + 1}{2^4} \\ &= \frac{5}{2^{10}} \end{aligned}$$

$\diamond$

The theory of $\mathcal{FP}$ provides a variety of built-in floating-point operations as defined in the IEEE standard 754-2008. This includes binary arithmetic operations (e.g. $+, -, \star, \div$), basic unary operations (e.g. $abs, -$), binary comparison operations (e.g. $\leq, <, \neq, =, >, \geq$), the remainder operation, the square root operation and more. Arithmetic operations are performed *as if with infinite precision*, but the result is then *rounded* to the "nearest" representable $\mathcal{FP}$ number according to the specified *rounding mode*. Five *rounding modes* are made available, as in [iee08].

The most common approach for $\mathcal{FP}$-satisfiability is to encode $\mathcal{FP}$ expressions into $\mathcal{BV}$ formulas based on the circuits used to implement floating-point operations, using appropriate under- and over-approximation schemes —or a mixture of both— to improve performance [BKW09, ZWR14, ZWR17, ZBWR18]. Then, the $\mathcal{BV}$-Solver is used to deal with the $\mathcal{FP}$ formula, using either the *eager* or the *lazy* $\mathcal{BV}$ approach. An alternative approach, based on *abstract interpretation*, is presented in [BDG$^+$13, BDG$^+$14, HGBK12]. With this technique, called *Abstract CDCL* (ACDCL), the set of feasible solutions is over-approximated with floating-point intervals, so that intervals-based conflict analysis is performed to decide $\mathcal{FP}$-satisfiability.

2.2.4 Combination of Theories in SMT

Typical SMT($\mathcal{T}$) applications deal with problems in which the theory $\mathcal{T}$ is given by the combination of two (or more) simpler theories, so that $\mathcal{T} \stackrel{\text{def}}{=} \bigcup_{i=1}^{n} \mathcal{T}_i$. For instance, an atom like $f(4x + y) = g(2x - y)$ combines both uninterpreted function symbols (i.e. f, g) with linear arithmetic constraints (i.e. $4x + y, 2x - y$). Modern "lazy" SMT solvers employ a variety of techniques for dealing with combination of theories.

When dealing with the combination of some stably infinite theories $\mathcal{T}_1$ and $\mathcal{T}_2$ with *disjoint signatures* (i.e. "Nelson-Oppen" theories), the Nelson-Oppen approach for theory combination [NO79, Opp80, Sho84] can be used. Two theories $\mathcal{T}_1$ and $\mathcal{T}_2$ are said to be *signature-disjoint* if $\mathcal{T}_1$ and $\mathcal{T}_2$ share no symbol other than the equality symbol. In the Nelson-Oppen schema, each $\mathcal{T}_i$-solver separately solves its own fragment of the input problem, limited to its own theory $\mathcal{T}_i$, and it is in direct communication with the other theory solver(s) to exchange implied equalities and disequalities over shared variables.

Other, more recent, approaches for theory combination include *Delayed Theory Combination* (DTC) and *Model-Based Theory Combination*.

In *Delayed Theory Combination*, [BBC$^+$05b, BBC$^+$06, BCF$^+$06], each $\mathcal{T}_i$-solver interacts only with the CDCL engine, and it does not directly exchange any information with the other

theory solvers. The CDCL engine is used to enumerate satisfiable propositional truth assignments that assign a value not only to the atoms in the input formula, but also to the interface (dis)equalities (i.e. (dis)equalities over variables shared by multiple theories). Moreover, the CDCL engine is also used to efficiently handle case splits resulting from the entailment of disjunctions of interface equalities in non-convex theories.

In *Model-Based Theory Combination*, [dMB08], interface equalities are built during the search using the model $\mathcal{M}_i$ generated by the corresponding $\mathcal{T}_i$-solver when a satisfiable (complete) truth assignment is found. More specifically, a new interface equality $u = v$ is generated for each pair of interface variables u and v such that $\mathcal{M}_i(u) = \mathcal{M}_i(v)$. Then, when the CDCL engine branches on an interface equality for the first time, it is initially assigned the value $\top$, so that its consistency with respect to other theories can be either confirmed or disproved; in the latter case it results in a $\mathcal{T}$-conflict and the search proceeds as usual.

In the case of theories that are not stably-infinite (e.g., the theory of Bit-Vectors), and thus are not Nelson-Oppen theories, other approaches have been proposed (see, e.g., [TZ05, RRZ05, JB10]).

2.3 Optimization Modulo Theories

As a first approximation, *Optimization Modulo Theories* (OMT) [NO06, CFG+10, ST12, DDMA12, MP13, CGSS13a, ST15a, LAK+14, LORR14] can be seen as the optimization-extended version of *Satisfiability Modulo Theories*. More in detail, given a satisfiable ground SMT formula φ and some objective function obj, *Optimization Modulo Theories* solves the problem of finding a model $\mathcal{M}$ of φ whose value of obj, denoted with $\mathsf{min}_{\mathsf{obj}}(\varphi)$, is minimum.

Due to its broad definition, *Optimization Modulo Theories* is an umbrella word that encompasses several —distinct, albeit related— problems and a variety of optimization techniques. Instead of following a simple chronological order, we chose to organize the OMT literature based on the specific class of OMT problem being targeted. The latter is determined based on which Theory the objective function obj belongs. As a result, we distinguish three main groups of works:

- Section §2.3.1 covers OMT($\mathcal{LRA} \cup \mathcal{T}$), dealing with *Linear Rational Arithmetic* ($\mathcal{LRA}$) cost functions
- Section §2.3.2 illustrates OMT($\mathcal{LIA} \cup \mathcal{T}$), dealing with *Linear Integer Arithmetic* ($\mathcal{LIA}$) cost functions
- Section §2.3.3 describes OMT($\mathcal{PB} \cup \mathcal{T}$), dealing with both *Pseudo-Boolean* ($\mathcal{PB}$) objectives, and also MAXSMT

Combination of Theories in OMT. One important result of [ST12, ST15a] is that the the same optimization techniques designed for OMT($\mathcal{LRA}$) can be used to deal with OMT($\mathcal{LRA} \cup \mathcal{T}$), where $\mathcal{T}$ is some stably-infinite theory (or a combination thereof) with equality such that $\mathcal{LRA}$ and $\mathcal{T}$ are signature-disjoint as in [NO79]. This result comes nearly for free when the underlying SMT solver uses *Delayed Theory Combination* [BBC+06] for dealing with multiple theories $\mathcal{T}$.

It is easy to see that the same result applies when the objective function is of any other type than $\mathcal{LRA}$ like, for example, a $\mathcal{LIA}$, a $\mathcal{LIRA}$, a $\mathcal{PB}$, a MAXSMT, a $\mathcal{BV}$ or a $\mathcal{FP}$ goal. Therefore, for the sake of a clear and readable explanation, in the following we illustrate the general optimization procedures and techniques for when $\mathcal{T}$ is the empty theory and refer the reader to [ST12, ST15a] for a detailed overview on how to handle the general case. For the same reasons, the same simplification is applied to any other variant of single- and multiple-objective OMT that is explored throughout this dissertation.

2.3.1 OMT ($\mathcal{LRA} \cup \mathcal{T}$)

The *Optimization Modulo Theories problem* for *Linear Rational Arithmetic* cost functions is defined as follows.

Definition 2.3.1. *(OMT($\mathcal{LRA} \cup \mathcal{T}$), OMT($\mathcal{LRA}$)). Let φ be a ground SMT($\mathcal{LRA} \cup \mathcal{T}$) formula and* obj *be a $\mathcal{LRA}$ variable occurring in φ. We call an Optimization Modulo $\mathcal{LRA} \cup \mathcal{T}$ problem, the problem of finding a model $\mathcal{M}$ for φ (if any) whose value of* obj*, denoted with* $\mathsf{min}_{\mathsf{obj}}(\varphi)$*, is minimum. We call an Optimization Modulo $\mathcal{LRA}$ problem, written OMT($\mathcal{LRA}$), an OMT($\mathcal{LRA} \cup \mathcal{T}$) problem where $\mathcal{T}$ is the empty theory. (The dual definition where we look for the maximum follows straightforwardly)*

As observed in [ST12], OMT($\mathcal{LRA} \cup \mathcal{T}$) allows for a straightforward encoding of various problem domains of interest, like *Linear Programming* (LP), *Linear Disjunctive Programming* (LDP) [Bal98] and *Linear Generalized Disjunctive Programming* (LGDP) [RG94].

Three main approaches have been proposed for dealing with OMT($\mathcal{LRA} \cup \mathcal{T}$). The first two are the *offline* and *inline schemata* presented in [ST12, ST15a] and implemented in OPTIMATHSAT. The third one is the *Symbolic Optimization Algorithm* of SYMBA, an OMT tool presented by Li et al. in [LAK+14].

Offline Schema [ST12, ST15a]

Figure 2.2 shows the mixed linear- and binary-search *offline schema* presented in [ST12, ST15a] for dealing with OMT($\mathcal{LRA} \cup \mathcal{T}$). In this approach, similarly to [LAK+14, BP14, BPF15], the

optimization search proceeds through a sequence of incremental calls to the underlying SMT solver, which is used as a black-box. Hereafter, we provide a concise description of this method based on the contents of [ST12, ST15a], and refer the reader to these publications for a more detailed presentation.

The algorithm takes as input a ground SMT($\mathcal{LRA} \cup \mathcal{T}$) formula φ, a $\mathcal{LRA}$ goal obj and an optional range $[l, u[$ for the optimization search. When not provided, l and u default to $-\infty$ and $+\infty$ respectively.

Remark 2.3.1. In [ST12, ST15a], the range $[\mathsf{lb}, \mathsf{ub}[$ is the domain of values for obj in which the OMT solver searches for a model $\mathcal{M}$ of φ when minimizing obj. After the first range update, the current upper bound ub corresponds to the value of obj in a known model $\mathcal{M}$ of φ. Therefore, the upper bound is excluded from the search domain to avoid generating the same model $\mathcal{M}$ again. Conversely, the lower bound lb is contained in the search interval because it is updated only when the conjunction $\varphi \wedge (\mathsf{obj} < \mathsf{lb})$ is found to be unsatisfiable. When obj is maximized the interpretation is dual and effective range is $]\mathsf{lb}, \mathsf{ub}]$.

The convention established in [ST12, ST15a] is that the same interpretation is applied to any user-provided value for the initial lower and upper bounds l and u, so that the initial range is $[l, u[$ in minimization and $]l, u]$ in maximization.

At the beginning, $\mathcal{M}$ is set to the $\emptyset$ and the initial range $[l, u[$ is used to initialize the current range $[\mathsf{lb}, \mathsf{ub}[$ (line 1). At any given point in time, the latter holds an over-approximation of the set of feasible values for obj. To ensure that this invariant holds when the search is started, φ is extended with a pair of constraints that bound the feasible domain of obj within the range $[\mathsf{lb}, \mathsf{ub}[$ (line 2).

The main optimization search is performed in a loop (lines 3-20), and terminates only when the $\mathsf{ub} \leq \mathsf{lb}$, that is, when the over-approximation of the feasible domain of obj becomes empty. Search progress is ensured, at each iteration, with an update to the value of either lb or ub that makes the range $[\mathsf{lb}, \mathsf{ub}[$ smaller. Each iteration of the main loop consists of a single *linear-* or *binary-search* step.

Remark 2.3.2. As observed in [ST12, ST15a], the underlying SMT solver is invoked *incrementally* at each iteration of the loop. Hence, subsequent calls to the SMT solver enjoy a substantial speedup from reusing lots of previously generated information like, for example, any clause created by the *clause learning* mechanism (see Section §2.2.2).

In a *linear-search* step, the code in the intervals of lines 5-8 and 14-19 is never executed. Hence, a *linear-search* step starts by searching for a truth assignment μ that propositionally

function OFFLINE_OMT(φ, obj, l, u)

1: $\mathsf{lb} \coloneqq l$; $\mathsf{ub} \coloneqq u$; $\mathcal{M} \coloneqq \emptyset$
2: $\varphi \coloneqq \varphi \cup \{\neg(\mathsf{obj} < \mathsf{lb}), (\mathsf{obj} < \mathsf{ub})\}$
3: **while** $(\mathsf{lb} < \mathsf{ub})$ **do**
4: $\quad binary\_step \coloneqq$ BINSEARCHSTEP()
5: $\quad$ **if** $(binary\_step)$ **then**
6: $\quad\quad \mathsf{pivot} \coloneqq$ COMPUTEPIVOT(lb, ub)
7: $\quad\quad PIV \coloneqq (\mathsf{obj} < \mathsf{pivot})$
8: $\quad\quad \varphi \coloneqq \varphi \cup \{PIV\}$
9: $\quad \langle res, \mu \rangle \coloneqq$ SMT.INCREMENTALSOLVE(φ)
10: $\quad$ **if** $(res == \text{SAT})$ **then**
11: $\quad\quad \langle \mathcal{M}, \mathsf{ub} \rangle \coloneqq \mathcal{LRA}$-MINIMIZE(obj, μ)
12: $\quad\quad \varphi \coloneqq \varphi \cup \{(\mathsf{obj} < \mathsf{ub})\}$
13: $\quad$ **else**
14: $\quad\quad$ **if** $(binary\_step)$ **then**
15: $\quad\quad\quad \eta \coloneqq$ SMT.EXTRACTUNSATCORE(φ)
16: $\quad\quad\quad$ **if** $(PIV \in \eta)$ **then**
17: $\quad\quad\quad\quad \mathsf{lb} \coloneqq \mathsf{pivot}$
18: $\quad\quad\quad\quad \varphi \coloneqq (\varphi \backslash \{PIV\}) \cup \{\neg PIV\}$
19: $\quad\quad\quad\quad$ **continue**
20: $\quad\quad \mathsf{lb} \coloneqq \mathsf{ub}$
21: **if** $(\mathcal{M} \neq \emptyset)$ **then**
22: $\quad$ **return** $\langle \text{SAT}, \mathsf{ub}, \mathcal{M} \rangle$
23: **else**
24: $\quad$ **return** $\langle \text{UNSAT}, +\infty, \emptyset \rangle$

Figure 2.2: Offline OMT($\mathcal{LRA}$) procedure based on Mixed Linear/Binary Search [ST12, ST15a].

satisfies φ (line 9). If φ is satisfiable, then the OMT solver invokes SMT.$\mathcal{LRA}$-MINIMIZE() to find the model $\mathcal{M}$ of minimum cost ub corresponding to the truth assignment μ (line 11). The function SMT.$\mathcal{LRA}$-MINIMIZE() is an extended version of the simplex-based $\mathcal{LRA}$-Solver of [DdM06b]. The minimum cost ub can be equal to $-\infty$ if obj is unbounded on the truth assignment μ. At this point, φ is extended with a unit clause of the form $(\mathsf{obj} < \mathsf{ub})$ (line 12), so that φ is no longer satisfied by μ. As a result, any future call to the underlying SMT solver is forced to look for a new propositional model μ' for which the value of obj can be smaller than ub. If instead the call to SMT.INCREMENTALSOLVE() at line 9 returns UNSAT, then it means that in the current search interval $[\mathsf{lb}, \mathsf{ub}[$ there is no valid assignment of value for obj. Hence, the lower bound lb is updated with the value of ub (line 20), causing the OMT solver to exit the loop the next time its guard is checked.

A *binary-search* step starts with a call to COMPUTEPIVOT() (line 6), an heuristic function that yields a new pivot value contained in the interval $[\mathsf{lb}, \mathsf{ub}[$ (e.g. $\frac{\mathsf{lb}+\mathsf{ub}}{2}$). Then the formula φ is extended with the (possibly new) atom PIV, defined as $(\mathsf{obj} < \mathsf{pivot})$, to *temporarily* restrict the feasible domain of obj in the interval $[\mathsf{lb}, \mathsf{pivot}[$ (lines 7-8). If the next call to SMT.INCREMENTALSOLVE() finds that φ is still satisfiable, then the OMT solver proceeds as in a *linear-search* step: it computes the new minimum ub and updates both $\mathcal{M}$ and φ. Otherwise, if φ is unsatisfiable, then the *unsatisfiable core* η is extracted and examined (lines 15-16). If the atom PIV belongs to η, then it means that there exists no satisfiable truth assignment μ for which obj can be assigned a value in the interval $[\mathsf{lb}, \mathsf{pivot}[$. Therefore, lb is set to the value of pivot and PIV is replaced by its negation inside φ so that the range $[\mathsf{pivot}, \mathsf{ub}[$ will be explored in the next iteration of the loop (lines 17-19). If instead $PIV \notin \eta$, then it means that $\varphi \setminus \{PIV\}$ is unsatisfiable for the whole range $[\mathsf{lb}, \mathsf{ub}[$. In this case, the search proceeds as in an unsatisfiable *linear-search* step by setting lb equal to ub and causing the loop to terminate.

When the optimization search has terminated (lines 21-24), the OMT solver yields a triple $\langle \text{SAT}, \mathsf{ub}, \mathcal{M} \rangle$ if φ was found to be satisfiable, and $\langle \text{UNSAT}, +\infty, \emptyset \rangle$ otherwise.

We report a few important facts about *binary-search* that were made in [ST12, ST15a].

First, a *binary-search* step can only be performed when both lb and ub have a finite value. In minimization a finite upper bound ub is easily determined by means of a *linear-search* step, therefore the user is only required to provide an initial lower bound. In maximization, the requirement is dual.

Second, a *binary-search* step must be interleaved infinitely often with a *linear-search* step to prevent non-termination when the non-empty range $[\mathsf{lb}, \mathsf{ub}[$ is unsatisfiable. In fact, executing exclusively in *binary-search* in this situation can result in an infinite number of lower bound updates, because infinite-precision arithmetic guarantees that it is always possible to find a new pivot in $[\mathsf{lb}, \mathsf{ub}[$, no matter how small the range becomes. On this regard, the authors of

[ST12, ST15a] suggested that, based on their empirical experience, the best performance is obtained by starting with an initial *linear-search* step that can provide tighter estimate of the upper bound ub, and then alternate each *binary-search* step with a *linear-search* one.

Third, a *binary-search* step is not necessarily more convenient than a *linear-search* step. On the one hand, limiting the search space to the interval $[\mathsf{lb}, \mathsf{pivot}[$ may result in a tighter upper bound ub than by searching it in the whole range $[\mathsf{lb}, \mathsf{ub}[$. On the other hand, this search can be very time-consuming if the interval $[\mathsf{lb}, \mathsf{pivot}[$ contains no satisfiable solution, because detecting $\mathcal{LIRA}$-unsatisfiability is typically much more expensive than generating a new model[4]. For this reason, the authors of [ST12, ST15a] proposed an *adaptive* version of BINSEARCHSTEP() that determines the next search step based on an heuristic evaluation of the cost-benefit ratio of the most recent *linear-* and *binary-* search steps.

Inline Schema [ST12, ST15a]

In addition to the *offline schema*, the authors of [ST12, ST15a] also presented in the same publications the so-called *inline schema*. Both approaches employ exactly the same range-minimization approach, comprised by a number of *linear-* and *binary-search* steps, that we have just described for the *offline schema*. However, while the *offline schema* proceeds through a sequence of incremental calls to the underlying SMT solver used as a black-box, in the *inline schema* the whole optimization search is pushed within the CDCL Boolean-search loop of the standard lazy SMT schema [Seb07, BSST09]. In this way, the optimum value of obj is retrieved with a single run of the SMT search-loop. The authors implemented both approaches in OPTIMATHSAT and reported consistently better performance with the *inline* architecture in the experimental evaluation of [ST12, ST15a].

Hereafter, we succintely describe the main aspects of the *inline schema* based on the content of [ST12, ST15a], bearing in mind the high-level of similarity with the already presented *offline schema*.

Initialization: the output model $\mathcal{M}$ is empty, while the current lower bound lb and the current-upper bound ub are respectively initialized to the input lower bound l and upper bound u if available, $-\infty$ and $+\infty$ otherwise.

Range Updating & Pivoting. The algorithm maintains the following invariant on the range $[\mathsf{lb}, \mathsf{ub}[$. The upper bound ub [resp. lower bound lb] is assigned the lowest [resp. highest] value v such that the atom $(\mathsf{obj} < v)$ [resp. $\neg(\mathsf{obj} < v)$] is assigned at level 0 of the CDCL Boolean-search loop. The search terminates if φ is unsatisfiable, so that there is no optimal solution,

[4]This conclusion is based on the empirical experience in [ST12, ST15a, ST15c] when dealing with OMT($\mathcal{LIRA} \cup \mathcal{T}$).

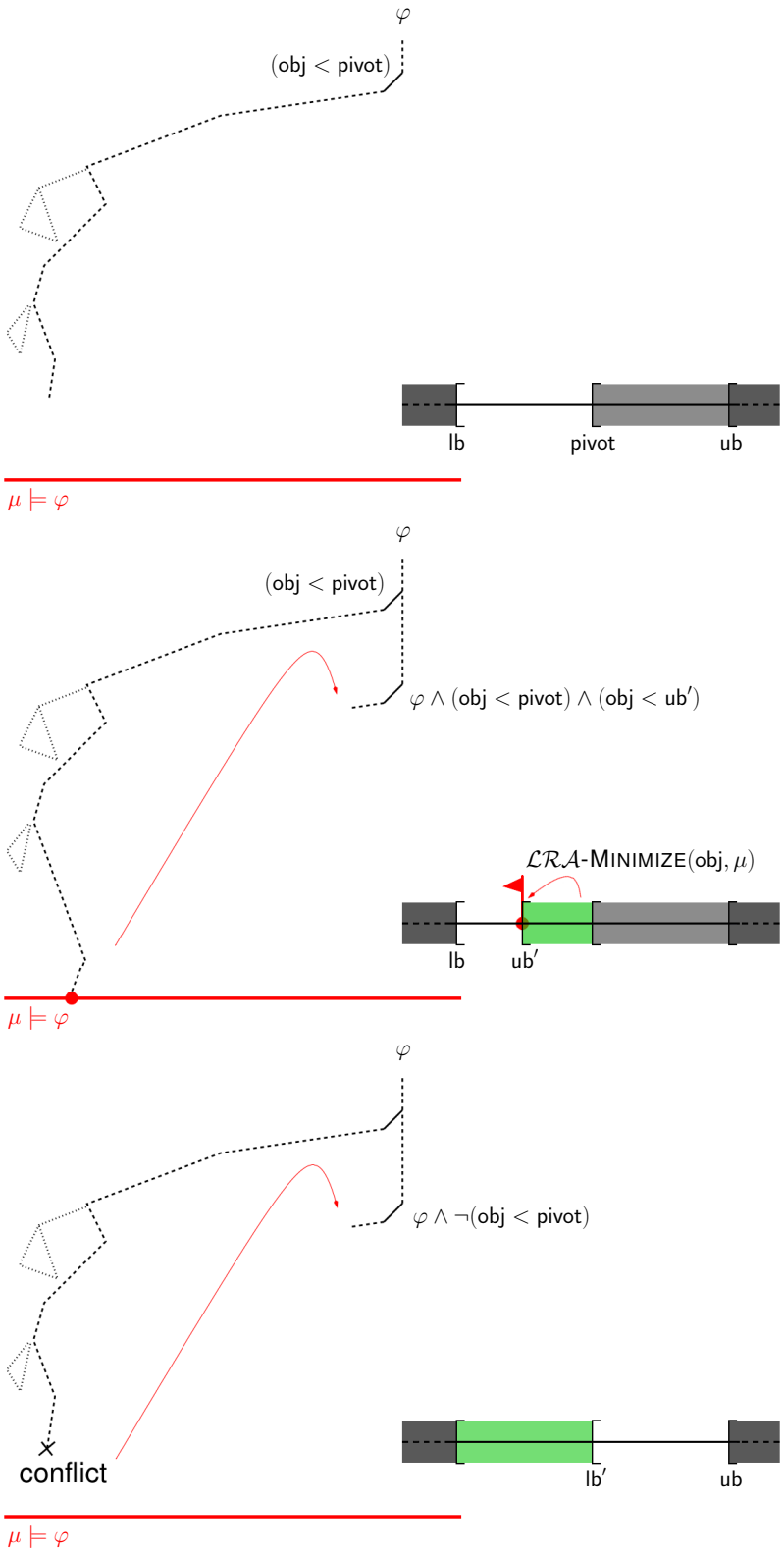


Figure 2.3: A possible execution of the *inline schema*. (I) Pivoting on $(\text{obj} < \text{pivot})$. (II) Decreasing the upper bound to ub'. (III) Increasing the lower bound to pivot.

or when ub becomes smaller or equal lb. Each time the search goes back to level 0, it can decide whether to execute the next search step in *linear-* or —when both lb and ub are finite— in *binary-search* mode. In the latter case, a $\mathsf{pivot} \in [\mathsf{lb}, \mathsf{ub}[$ value is computed (e.g. with $\frac{\mathsf{lb}+\mathsf{ub}}{2}$), and the (possibly new) atom $PIV = (\mathsf{obj} < \mathsf{pivot})$ is forcibly decided at level 1 of the CDCL Boolean-search loop. This decision temporarily restricts the search to the interval $[\mathsf{lb}, \mathsf{pivot}[$, as depicted figure 2.3-(I).

Decreasing the Upper Bound. Whenever the CDCL search generates a complete truth assignment μ that propositionally satisfies φ, the OMT solver computes ub', that is, the minimum value of obj in correspondence with the truth assignment μ. This is done by $\mathcal{LRA}$-MINIMIZE(), that is incrementally called after the decision procedure for $\mathcal{LRA}$-satisfiability to avoid starting from scratch. Then, the unit clause $(\mathsf{obj} < \mathsf{ub}')$ is learned, so that the CDCL Boolean-search is forced to backjump to level 0 and unit-propagate it. This permanently restricts the search to the interval $[\mathsf{lb}, \mathsf{ub}'[$, as depicted figure 2.3-(II). When the upper bound decrement occurs in a *binary-search* step, the atom $(\mathsf{obj} < \mathsf{pivot})$ is also learned prior to backjumping to level 0. This allows the CDCL loop to reuse any clause of the form $\neg(\mathsf{obj} < \mathsf{pivot}) \vee C$ that might have been generated when searching for a cost in the range $[\mathsf{lb}, \mathsf{pivot}[$.

Increasing the Lower Bound. When $\varphi \wedge \{(\mathsf{obj} < \mathsf{pivot})\}$ is $\mathcal{LRA}$-inconsistent, the CDCL Boolean-search eventually generates a conflict clause of the form $\neg(\mathsf{obj} < \mathsf{pivot}) \vee \eta'$ such that all literals in η' are permanently assigned to $\top$ at level 0. As a result, the search is forced to backjump at level 0 and unit-propagate $\neg(\mathsf{obj} < \mathsf{pivot})$. This case, shown in Figure 2.3-(III), permanently restricts the cost range to the interval $[\mathsf{pivot}, \mathsf{ub}[$.

We refer the reader to the description of the *offline schema* for what concerns some key aspects of *binary-search* in the *inline schema*, since the same considerations apply here.

Symbolic Optimization Algorithm [LAK$^+$14]

In [LAK$^+$14], Li et al. proposed SYMBA, an alternative OMT($\mathcal{LRA} \cup \mathcal{T}$) tool built on top of the Z3 SMT solver.

Differently than [ST12, ST15a], the authors deal with the optimization of multiple *independent* $\mathcal{LRA}$ objectives $\mathsf{obj}_1, ..., \mathsf{obj}_N$ at the same time, where the goal is to find the set of models $\mathcal{M}_1, ..., \mathcal{M}_N$ such that each $\mathcal{M}_i$ is optimal with respect to its corresponding goal obj_i. To this aim, dealing with this problem can be seen as being totally equivalent to solving N single-objective OMT($\mathcal{LRA} \cup \mathcal{T}$) problems, one for each obj_i. However, handling multiple objectives at the same time allows for sharing SMT search steps among multiple objectives and a better exploitation of clause learning, as shown in the performance evaluations of [LAK$^+$14, ST15c].

Li et al. presented both and "offline" and an "inline" version of their tool. In the "offline"

implementation, the underlying SMT solver is used as a black-box, and the optimization search is advanced via the incremental application of a set of inference rules that either (1) force an improvement of the current solution along some objective direction (2) prove that some objective is unbounded. The "inline" implementation extends the "offline" version by modifying the $\mathcal{LRA}$-Solver in Z3 to also return the optimal value of a goal obj_i given a fixed truth assignment μ, as with the *inline schema* of [ST12, ST15a]. As shown in the empirical evaluation of [LAK+14], this addition allows the "inline" approach to outperform the "offline" architecture since it drastically reduces the number of SMT queries necessary to find the optimal solution.

2.3.2 OMT ($\mathcal{LIA} \cup \mathcal{T}$)

The *Optimization Modulo Theories problem* for *Linear Integer Arithmetic* cost functions is defined as follows.

Definition 2.3.2. *(OMT($\mathcal{LIA} \cup \mathcal{T}$), OMT($\mathcal{LIA}$)). Let φ be a ground SMT($\mathcal{LIA} \cup \mathcal{T}$) formula and* obj *be a $\mathcal{LIA}$ variable occurring in φ. We call an Optimization Modulo $\mathcal{LIA} \cup \mathcal{T}$ problem, the problem of finding a model $\mathcal{M}$ for φ (if any) whose value of* obj*, denoted with $\mathsf{min}_{\mathsf{obj}}(\varphi)$, is minimum. We call an Optimization Modulo $\mathcal{LIA}$ problem, written OMT($\mathcal{LIA}$), an OMT($\mathcal{LIA} \cup \mathcal{T}$) problem where $\mathcal{T}$ is the empty theory. (The dual definition where we look for the maximum follows straightforwardly)*

To the best of our knowledge, there exists only one publicly available work on OMT($\mathcal{LIA}$)[5] and this is the master's thesis of R. O. Vendrell [Roc11], which builds upon the "SMT with progressively stronger theories" approach of [NO06] (see Section §2.3.3).

In this thesis, the authors extended the BCLT SMT solver [BNO+08] with a minimization procedure for $\mathcal{LIA}$ objectives embedded within the $\mathcal{T}$-Solver for *Linear Integer Arithmetic*[6]. As a first building step, the authors provide a detailed description of how the decision procedure for $\mathcal{LRA}$-satisfiability —based on the simplex-based $\mathcal{LRA}$-Solver of [DdM06b]— can be extended with optimization capabilities. The end result of this transformation is similar to the function $\mathcal{LRA}$-MINIMIZE() described in [ST12, ST15a] (see Section §2.3.1) for

[5] We recall that in this section we consider only those works that were made publicly available prior to the start of this Ph.D. (November, 2014). Any other work is reported in the Related Work (§3). Moreover, here we do not consider those approaches that deal with the more-specific OMT($\mathcal{PB} \cup \mathcal{T}$) and MAXSMT problems with Integer weights. These approaches are separately covered in Section §2.3.3.

[6] To be precise, [Roc11] also describes another implementation based on using an off-the-shelf ILP solver (CPLEX) for the optimization part. This approach is not considered here because the experimental evaluation of [Roc11] has shown that, on the benchmark-set being considered, the combination of BCLT with CPLEX did not show any added benefit compared with using CPLEX as a standalone solver. We refer the interested reader to [Roc11] for more details.

OMT($\mathcal{LRA} \cup \mathcal{T}$). Given a complete truth assignment μ and an objective obj, it returns the model interpretation $\mathcal{M}$ of μ that makes obj $\mathcal{LRA}$-minimum. As stated in [Roc11], if some variable need to take Integer values, then a complete $\mathcal{LIA}$-MINIMIZE() procedure can be obtained by combining $\mathcal{LRA}$-MINIMIZE() with the well-known Branch&Bound and cutting planes techniques.

The overall optimization search is based on [NO06] and it proceeds similarly to the *inline schema* —running in *linear-search* mode— of [ST12, ST15a] and described in Section §2.3.1. Given a complete truth assignment μ that satisfies the input formula φ, the solver invokes the $\mathcal{LIA}$-MINIMIZE() procedure to retrieve the corresponding minimum value ub of obj, and then adds the *linear* cut obj $<$ ub to the database of constraints, so that the SMT solver is forced to find a new truth assignment μ' improving the value of obj. The search ends when the value of obj cannot be improved any further.

2.3.3 OMT($\mathcal{PB} \cup \mathcal{T}$)/MAXSMT

Two important subcases of OMT($\mathcal{LRA} \cup \mathcal{T}$) are represented by OMT($\mathcal{PB} \cup \mathcal{T}$) and MAXSMT[7]. In this section, we present both subcases together since they allow to deal with the same class of problems: an instance of MAXSMT can be encoded as a OMT($\mathcal{PB} \cup \mathcal{T}$) problem and vice-versa.

Another important aspect to be noticed is that in both cases the optimization search has only a Boolean component, due to the fact that the value of the cost function is univocally determined by the truth assignment μ over the atoms of the input formula. This is substantially different from the general case of OMT($\mathcal{LRA} \cup \mathcal{T}$), for which it is necessary to compute the minimum-cost $\mathcal{LRA}$-model for each satisfying truth assignment μ found along the optimization search.

OMT with Pseudo-Boolean cost functions

The *Optimization Modulo Theories problem* for *Pseudo-Boolean* cost functions is defined as follows.

[7]Here, we immediately note that MAXSMT is MAXSAT lifted to the case of SMT formulas. In this thesis, we assume that the reader is familiar with MAXSAT, and otherwise refer to [LM09, MSAGL11] for an introduction on the topic.

Definition 2.3.3. *(OMT($\mathcal{PB} \cup \mathcal{T}$), OMT($\mathcal{PB}$)). Let φ be a ground SMT($\mathcal{T}$) formula and* obj *be defined as follows:*

$$\mathsf{obj} \stackrel{def}{=} \sum_i w_i \cdot A_i \ , \ w_i \in \mathbb{R} \text{ and } A_i \text{ are Boolean variables}$$

We call an Optimization Modulo $\mathcal{PB} \cup \mathcal{T}$ problem, the problem of finding a model $\mathcal{M}$ for φ (if any) whose value of obj*, denoted with* $\mathsf{min}_{\mathsf{obj}}(\varphi)$*, is minimum. We call an Optimization Modulo $\mathcal{PB}$ problem, written OMT($\mathcal{PB}$), an OMT($\mathcal{PB} \cup \mathcal{T}$) problem where $\mathcal{T}$ is the empty theory.*

In the literature, two main approaches have been proposed to deal with OMT($\mathcal{PB}$): an encoding of the problem into OMT($\mathcal{LRA}$) [ST12, ST15a] and, when all weights $w_i \in \mathbb{Z}$, the "Theory of Costs" $\mathcal{C}$ [CFG$^+$10].

OMT($\mathcal{LRA}$) encoding [ST12, ST15a]. A OMT($\mathcal{PB} \cup \mathcal{T}$) problem $\langle \varphi, \mathsf{obj} \rangle$ can be encoded as an OMT($\mathcal{LRA}$) instance $\langle \varphi', \mathsf{obj}' \rangle$ as follows. First, a fresh Rational variable x_i is introduced for each w_i in obj. Then, φ' and obj$'$ are defined as

$$\varphi' \stackrel{\text{def}}{=} \varphi \wedge \bigwedge_i ((\neg A_i \vee (x_i = w_i)) \wedge (A_i \vee (x_i = 0))) \wedge \tag{2.2a}$$

$$\bigwedge_i ((0 \leq x_i) \wedge (x_i \leq w_i)) \tag{2.2b}$$

$$\mathsf{obj}' \stackrel{\text{def}}{=} \sum_i x_i \tag{2.2c}$$

As noted in [ST15a], although the constraints in (2.2b) may appear redundant from a logical perspective, they noticeably improve the performance. This is because these constraints allow early-pruning calls to the $\mathcal{LRA}$-Solver to detect a possible $\mathcal{LRA}$ inconsistency among the current partial truth assignment over variables A_i and any linear cut of the form $\neg(\mathsf{ub} \leq \mathsf{obj})$, that is typically pushed on the formula stack during the minimization of obj.

Theory of Costs [CFG$^+$10]. In [CFG$^+$10], Cimatti et al. introduced the "Theory of Costs" $\mathcal{C}$ and extended the standard lazy SMT schema with a decision procedure for $\mathcal{C}$, called $\mathcal{C}$-Solver, that would deal with both Pseudo-Boolean ($\mathcal{PB}$) objectives and Pseudo-Boolean constraints. The proposed $\mathcal{C}$-Solver was combined with two distinct single-objective optimization schemas, one based on branch-and-bound and another on bisection-search, and implemented in a fork of the MATHSAT [matb] SMT solver.

The "Theory of Costs" $\mathcal{C}$ allows for the definition of multiple objectives obj^j, each taking the form

$$\mathsf{obj}^j \stackrel{\text{def}}{=} \sum_{i=1}^{N_j} \text{ITE}(A_i^j, w_i^j, 0) \tag{2.3}$$

where ITE is a function returning w_i^j when A_i^j is assigned to true and 0 otherwise. A fresh Integer variable c^j is introduced to represent the value of each obj^j. Moreover, two additional predicates are introduced with the language of theory $\mathcal{C}$. The first is a binary predicate $\text{BC}(c^j, w)$, requiring c^j to be smaller or equal to the integer constant w. The second is a ternary predicate $\text{IC}(c^j, i, w_i^j)$, that states that the i-th component of the pseudo-boolean sum (2.3) increases the value of c^j by an amount equal to w_i^j. These predicates are used to encode the objective function, that is thus kept signature-disjoint with respect to the other theories $\mathcal{T}$ appearing in the problem.

Notice that, while the language of theory $\mathcal{C}$ allows one to define multiple objective functions, the optimization procedures in [CFG+10] can handle the optimization of only one obj^j goal at a time. In addition, these routines are limited to deal with $\mathcal{PB}$ objectives with Integer weights only.

Partial Weighted MAXSMT

The *Partial Weighted* MAXSMT *problem* is defined as follows.

Definition 2.3.4. *(Partial Weighted* MAXSMT*, Partial* MAXSMT*, Weighted* MAXSMT*). A Partial Weighted* MAXSMT *problem is a pair $\langle \varphi_h, \varphi_s \rangle$, where φ_h is the set of "hard" $\mathcal{T}$-clauses, and φ_s is a collection of positive-weighted "soft" $\mathcal{T}$-clauses of the form $\langle C_i, w_i \rangle$, and the goal is to find the maximum-weight set of $\mathcal{T}$-clauses ψ_s, $\psi_s \subseteq \varphi_s$, such that $\varphi_h \cup \psi_s$ is $\mathcal{T}$-satisfiable [NO06, CFG+10, ABP+11b, CGSS13a].*

A Partial MAXSMT *problem is a Partial Weighted* MAXSMT *problem in which all "soft" $\mathcal{T}$-clauses in φ_s have a unitary weight.*

A Weighted MAXSMT *problem is a Partial Weighted* MAXSMT *problem in which the set of "hard" $\mathcal{T}$-clauses φ_h is empty.*

In the literature, three main approaches have been presented to deal with partial weighted MAXSMT problems. The first is "SMT with progressively stronger theories", proposed in [NO06]; the second is to encode MAXSMT as an OMT($\mathcal{PB}$) or OMT($\mathcal{LRA}$) instance [ST12, ST15a], and the third is to combine a MAXSAT solver and an SMT solver together [CGSS13a, BP14, BPF15].

SMT with progressively stronger theories [NO06]. In [NO06], Nieuwenhuis and Oliveras presented the general "SMT with progressively stronger theories" framework for dealing with weighted MAXSAT and weighted MAXSMT problems, that has been implemented in BCLT [BNO+08].

Given a MAXSMT pair $\langle \varphi_h, \varphi_s \rangle$, this approach introduces a fresh Boolean variable p_i and a fresh Integer variable k_i for each "soft" clause $\langle C_i, w_i \rangle$, so that each clause C_i is replaced by $C_i \vee p_i$. Then, an initial background theory $\mathcal{T}_0$ is introduced and given the constraints $\bigwedge_{i=1}^{N}(p_i \rightarrow (k_i = w_i)) \wedge (\neg p_i \rightarrow (k_i = 0))$, plus an additional cost function $k_1 + ... + k_N \leq u$, where u is a fresh Integer variable. In addition, the underlying SMT solver is extended with a new DPLL ($\mathcal{T}$) rule $\mathcal{T}$-STRENGTHEN that allows one to iteratively *strengthen* the initial theory $\mathcal{T}_0$ via Branch&Bound up until the optimal solution is found.

Each time a complete truth assignment μ_i is found, such that μ propositionally satisfies the initial set of constraints and it is $\mathcal{T}_i$-consistent, the cost function is evaluated and the corresponding value ub_i becomes the new upper bound of the optimization search. Then, the application of $\mathcal{T}$-STRENGTHEN replaces the theory $\mathcal{T}_i$ with a theory $\mathcal{T}_{i+1}$ that extends $\mathcal{T}_i$ with a linear cut of the form $u < \mathsf{ub}_i$. As a result, any truth assignment μ' for which the value of the cost function is not smaller than ub_i is now $\mathcal{T}_{i+1}$-inconsistent. Consequently, the rule application causes the underlying CDCL engine to backjump and look for a novel truth assignment μ' that may improve the value of the cost function. The optimization search terminates when no such truth assignment is found.

OMT($\mathcal{PB}$)/OMT($\mathcal{LRA}$) encoding [ST12, ST15a]. A MAXSMT pair $\langle \varphi_h, \varphi_s \rangle$ can be encoded into a OMT($\mathcal{PB}$) pair $\langle \varphi, \mathsf{obj} \rangle$ as follows. First, a fresh Boolean variable A_i is introduced for each soft constraint $C_i \in \varphi_s$. Then, φ and obj are defined as

$$\varphi \stackrel{\text{def}}{=} \varphi_h \cup \bigcup_{\langle C_i, w_i \rangle \in \varphi_s} \{(A_i \vee C_i)\}; \tag{2.4a}$$

$$\mathsf{obj} \stackrel{\text{def}}{=} \sum_{\langle C_i, w_i \rangle \in \varphi_s} w_i \cdot A_i \tag{2.4b}$$

The resulting OMT($\mathcal{PB}$) instance can then be directly solved with any technique that directly targets this kind of encoding (e.g. [CFG+10]), or subdue a subsequent transformation step into OMT($\mathcal{LRA}$) using equations (2.2), so that any OMT($\mathcal{LRA}$) solver can be employed.

MAXSAT and SMT combination [ABP+11b, CGSS13a, BP14, BPF15]. With this approach, some dedicated MAXSAT engine is coupled with an SMT solver, so that the former is leveraged to find the Boolean abstraction with minimum cost and the second provides the

$\mathcal{T}$-solvers for deciding the $\mathcal{T}$-satisfiability of any Boolean assignment found by the MAXSAT engine.

Ansótegui et al. [ABP+11b] describe an experimental evaluation over *Resource-Constrained Project Scheduling Problem* (RCPSP) instances using an implementation of a MAXSMT solver, built on top of YICES. The underlying SMT solver was extended with two MAXSAT algorithms based on unsat-core extraction, WPM1 [ABL09] and WBO [MSP09], enriched with an heuristic giving priority to cores involving constraints with higher weights. This implementation is not publicly available.

In [CGSS13a], Cimatti et al. presented a "modular" version of this approach, that allows one to combine a modern lazy SMT solver with an arbitrary propositional MAXSAT solver, used as a black-box. In this architecture, the SMT solver is used to produce an increasingly larger set of theory lemmas, whose Boolean abstraction is then fed to the MAXSAT engine to progressively refine the sequence of solutions that are found by it. The search terminates with an optimal solution when the truth assignment μ generated by the MAXSAT engine does not result in any $\mathcal{T}$-conflict within the SMT solver, or when a conflict involving only "hard" clauses is generated, meaning that the input problem is unsatisfiable as a whole. The authors of [CGSS13a] implemented their solution on top of the MATHSAT5 SMT solver, and combined it with a number of external MAXSAT engines.

In most situations, the use of a dedicated MAXSAT engine can outperform the translation into OMT($\mathcal{PB}$)/OMT($\mathcal{LRA}$) by a significant margin. However, there are some situations in which the encoding into OMT($\mathcal{PB}$)/OMT($\mathcal{LRA}$) is the only applicable approach in practice due to some inherent limitations of the former approach. The first issue is that, to the best of our knowledge, many MAXSAT engines deal with integer weights only and, unlike OMT, some of them might suffer when dealing with problems containing large and non factorable weights. Both of these conditions apply, for example, to the OMT formulas used to deal with Support Vector Machines in [TSP17]. In this case, the weight of a "soft" clause is a high-precision rational value resulting from previous runs of the Machine Learning approach, and rounding these values would affect the accuracy of the whole learning process. The second issue is that a dedicated MAXSAT engine cannot be used in the presence of OMT problems featuring an objective function that is the result of the linear combination of Pseudo-Boolean and arithmetic terms (like, e.g., for Linear Generalized Disjunctive Programming problems [ST15a]), or the non-trivial combination of several Pseudo-Boolean sums as in [TSP17].

Generalized MAXSMT

The *Partial Weighted* MAXSMT *problem* can be generalized as follows.

Definition 2.3.5. *(Generalized* MAXSMT*). A Generalized* MAXSMT *is a Partial Weighted* MAXSMT *in which the weights w_i are not restricted to be positive.*

Intuitively, in a *Generalized* MAXSMT, a negative weight w_i becomes a reward —instead of a cost— for falsifying the corresponding clause C_i.

The solution of a *Generalized* MAXSMT $\langle \varphi_h, \varphi_s \rangle$ can be found with the aid of any MAXSMT solver upon applying the following transformations. First, any zero-weighted clause in φ_s is removed from the problem as it provides no contribution to the solution. Then, any soft clause $\langle C_i, w_i \rangle$ such that $w_i < 0$ is replaced by a new soft clause $\langle \neg C_i, -w_i \rangle$. As a result of this transformation, the optimal model $\mathcal{M}$ of the new MAXSMT problem $\langle \varphi_h, \varphi'_s \rangle$, extended with a suitable Boolean assignment for any zero-weighted soft clause that was previously removed, is also an optimal model for the original *Generalized* MAXSMT problem. In fact, the only difference among the two encodings lies in the numerical value of the cumulative-weight of all unsatisfied soft clauses, that we also call the objective function. More precisely, the optimal value of objective in the *Generalized* MAXSMT instance is equal to that of the MAXSMT encoding plus $\sum_{\langle C_i, w_i \rangle \in \varphi_s | w_i < 0} w_i$.

Any *Generalized* MAXSMT problem is encoded as an OMT($\mathcal{PB}$) instance with the same approach used for MAXSMT. In addition, any OMT($\mathcal{PB}$) instance $\langle \varphi, \mathsf{obj} \rangle$, where $\mathsf{obj} \stackrel{\text{def}}{=} \sum_i w_i \cdot A_i$, can be rewritten as a *Generalized* MAXSMT instance:

$$\langle \varphi, \bigcup_i \langle \neg A_i, w_i \rangle \rangle \tag{2.5}$$

Clearly, when $w_i > 0$ for every i, then OMT($\mathcal{PB}$) maps directly into MAXSMT.

From now on, unless differently specified, we will use MAXSMT to denote the general case of *Generalized Partial Weighted* MAXSMT.

2.4 Constraint Programming and SAT/SMT/OMT

In this section, we briefly introduce *Finite Domain Constraint Programming* (FDCP), a restriction of *Constraint Programming* (CP) in which the domain of unknown variables is always finite. Since we are mainly interested in talking about the mutual connections among FDCP and SAT, SMT and OMT solving, we do not provide a comprehensive background on FDCP and touch only a few of its most essential aspects. For a more in-depth and detailed introduction to the topic, we refer the interested reader to publications on the topic such as [Tsa93, Bar99, RBW06], that are also the main sources of the material being presented.

2.4.1 (Finite Domain) Constraint Programming

At a high-level, *Constraint Programming* (CP) can be seen as a programming paradigm for solving large, combinatorial, problems described in terms of a list of high-level constraints expressing the relationship among several variables, each of which takes a value in a given domain.

Definition 2.4.1. *(Constraint Satisfaction Problem [Tsa93, RBW06]). More formally, a* Constraint Satisfaction Problem *(*CSP*) is defined as:*

- *a set of* variables $X = \{x_1, ..., x_n\}$,
- *for each variable x_i, a finite domain D_i of possible values, and*
- *a set of* constraints *that restrict the values that the variables can simultaneously take.*

CSP problems are most commonly solved either by *systematic search*, usually enhanced with *Constraint Propagation* techniques for greater efficiency, or by *stochastic and heuristic search*.

In *systematic search*, the space of solutions is traversed by extending a partial assignment of values over the variables x_i, until a consistent solution is found. For better performance, *systematic search* is complemented with conflict analysis. This is used to guide a backjumping mechanism that rolls back the search, undoing the assignment decision causing the inconsistency [Gas79]. Alternatively, instead of conflict analysis, the search can use a look-ahead schema. In this approach, the domain of the variables that are not already contained in the current assignment is (temporarily) restricted along the search to exclude any future conflict with the assignment under construction [Bar99].

Systematic search is often combined with *consistency techniques* [Mon74, Wal75, Mac77, Kum92], that analyze the constraint graph of a CSP problem to remove (some) inconsistent values from the domain of variables. In the constraint graph, nodes and edges correspond to the variables and constraints of the CSP problem respectively. Inconsistent values can be identified by analyzing either unary constraints, as in Node Consistency (NC), or binary constraints using the notion of arc-consistency, [Lar02, DAC10], that allows one to discard any value of a variable domain for which another variable does not have a corresponding satisfiable value.

In *stochastic and heuristic search* —like, e.g., in the *hill-climbing* algorithm [Nil80] or the *min-conflicts* heuristic [MJPL92]— a complete assignment is first randomly generated and then refined via minor value adjustments that reduce the number of conflicting constraints. Random restarts and heuristic techniques are employed to escape from local minima.

When a CSP is paired with an objective function, which is defined over a subset of the variables in X, we speak of *Constraint Optimization Problem* (COP) [Tsa93, RBW06]. Intuitively, the distinction among a CSP and a COP is that the former is only interested in finding a generic solution (or all solutions) of a problem, while the latter looks for the optimal solution (or a good approximation of the optimal solution) according to some fixed function ranking all possible solutions. The most widely known approach for dealing with optimization in CSP is *Branch and Bound* [LW66]. In the simplest implementation, the search space is explored systematically, (possibly) enumerating all (satisfiable) assignment of values. In minimization, each time the current assignment is extended with a new value, an heuristic function is evaluated. If the result of this evaluation exceeds a given cutoff threshold —that is initially set to $+\infty$ and updated each time a complete assignment is generated— then current assignment is discarded, along all of its possible extensions, and the search is rolled back to a previous decision level in which a different value assignment can be performed.

2.4.2 MINIZINC

MINIZINC, [NSB+07], is a widely adopted high-level declarative language for modeling CSP problems. For a detailed presentation of this language, we refer the interested reader to [Minb]. For the purposes of this dissertation, it suffices to know that the MINIZINC standard, [mina],

(I) defines three scalar types (Booleans, machine Integers and Floats) and two compound types (sets and fixed-size arrays of some scalar type),

(II) provides an extensive library of predefined *global constraints*, that increase both the easiness of use and the readability of MINIZINC models,

(III) supports useful language constructs such as `if-then-else`, `let` expressions, universal and existential comprehensions over finite domains, user-defined predicates and more.

A MINIZINC model is typically flattened into a FLATZINC [fla] instance, using the MZN2FZN compiler, before it is handed over to a MINIZINC solver. The purpose of FLATZINC is to bridge the gap among the high-level modeling in MINIZINC, and the need for a fixed, and easy-to-parse, input format that simplifies the implementation of the input interface of a MINIZINC solver. To this aim, we notice that the *global constraints* in MINIZINC express more complex relations among the objects of the language than those FLATZINC constraints. Normally, a MINIZINC solver is not required to directly support any *global constraints*, as these can be compiled by the MZN2FZN tool into a standardized FLATZINC representation that uses only

regular constraints and, if necessary, a number of fresh support variables. Even so, it can be convenient for a MINIZINC solver to handle *global constraints* directly, especially when it can use *ad hoc* decision procedures for dealing with them efficiently.

Each year, the CP community gravitating around the MINIZINC language hosts the MINIZINC Challenge, [Minb], a competition among MINIZINC solvers on a vast library of benchmarks containing planning, scheduling and logistic problems (and more).

2.4.3 FDCP vs. SAT, SMT and OMT

For many years, the research on FDCP and that on SAT—including its recent SMT and OMT extensions— has proceeded on parallel tracks, often with a significant degree of exchange of good ideas and techniques. Originally, this separation can be attributed to the different focus of their respective research communities, with the former more preoccupied about solving planning, scheduling and logistic problems, and the latter focused on the formal verification of both Software and Hardware systems. Over the years, the cross-fertilization among the two fields has pushed the two communities increasingly close to one another. Nowadays, the solvers designed in each, respective, community not only adopt similar techniques but they can also sometimes be used to deal with similar (or even coinciding) applications, at least for some classes of problems (e.g. planning and scheduling).

Distinctive features.

Despite some overlap, there still remain significant differences among the two worlds.

FDCP solvers typically display a stronger focus on combinatorial reasoning over finite domains (e.g. integers) than SMT and OMT solvers. To this aim, they benefit from very efficient *consistency algorithms* that reduce the number of possible conflicts encountered along the search. In addition, FDCP applications are typically modeled in widely adopted, high-level declarative languages such as the MINIZINC format in Section §2.4.2. The latter provides a variety of standardized *global constraints* for modeling complex subproblems recurring in many applications, so that *ad hoc* procedures can be implemented to efficiently deal with them.

SAT, SMT and OMT solvers —taken altogether— typically provide very efficient Boolean reasoning capabilities and an incremental interface that allows one to reuse learned information to increase the speed of subsequent searches when checking the satisfiability of closely related instances. Moreover, *lazy* SMT and OMT solvers (see Section §2.2.1) commonly display a greater level of expressiveness than FDCP tools, at least in terms of supported theories. For instance, SMT and OMT solvers support the theory of arrays ($\mathcal{AR}$), the theory of uninterpreted

functions with equality ($\mathcal{EUF}$), and several other theories, also with infinite-domain, as those described in Section §2.2.3.

When focusing only on *linear arithmetic*, which is handled by both categories of solvers, we notice that an SMT (and OMT) solver must provide —among other features— accurate infinite precision reasoning (at least as a fall-back), the ability to extract a (possibly small) subset of conflicting constraints from an unsatisfiable formula —the so-called unsat core— and also efficient procedures for the typical incremental usage of the tool (see Section §2.2.2). In contrast, the typical MINIZINC solver is not required to support any of these features, so that it can use more sophisticated techniques for *linear arithmetic* than those typically implemented in SMT and OMT solvers.

The most widely adopted language to model SMT and OMT applications is represented by the SMT-LIBv2 format, [smt], (and its extensions) which is a much simpler and lower-level language than those used by FDCP solvers. In particular, due to the lack of *global constraints*, complex subproblems recurring in many applications have to be manually broken down and encoded in terms of simpler grammatical structures (each time). However, in SMT and OMT the same problem can often be modeled in multiple ways, using a different subset of theories or constraints. On this regard, we notice that the particular choice of theories and constraints used to model a certain problem can have a huge impact on the performance of SMT and OMT solvers. This can be due to multiple reasons such as the different complexity of each theory, the efficiency of the decision procedures implemented in the corresponding $\mathcal{T}$*-solver*, the complex and hardly-predictable interaction between the input set of constraints and the various heuristics implemented in SMT and OMT solvers. As a result, modeling problems for SMT and OMT solvers is typically harder than doing so for FDCP solvers, and it requires a higher level of expertise to guarantee the best performance —and, in some cases, even the ability to produce a definitive answer— of the SMT and OMT tool used to solve it.

Related Research.

Recently, there has been an increasing interest in bridging the gap among the two research communities and, in particular, compare the effectiveness of FDCP, SAT, SMT and OMT tools on problems steaming from the other research community. In fact, two of the relevant challenges in Satisfiabiliy Modulo Theories characterized in [NO06, NORCR07] are 1) integrating SMT with techniques used in *Constraint Programming* to deal with *global constraints* and 2) find new solutions to tackle *Partial Weighted* MAXSMT to handle *Weighted* CSP problems efficiently. This research goal extends also to other research communities such as *Mixed Integer Linear Programming* (MILP), an extension of *Linear Programming (LP)* that involves both discrete

and continuous variables.

CSP and FDCP. In [BPV09, BPSV09], Bofill et al. presented SIMPLY, a compiler that translates CSP problems encoded in a declarative language similar to MINIZINC, [Minb], into SMT-LIBV2, [smt], the standard input format of SMT solvers. In a follow up of that work, Bofill et al. presented FZN2SMT [BSV10, BPSV12], a novel framework for translating CSP problems from the MINIZINC format into Version 1 of the SMT-LIB language. Differently than SIMPLY, the new framework is comprised of an existing MZN2FZN tool, that compiles the original models into the FLATZINC format, and a novel FZN2SMT compiler performing the last step of the conversion. Optimization problems, that cannot be encoded in the standardized SMT-LIB format, are solved by the FZN2SMT compiler directly, using an optimization procedure built on top of YICES, [DdM06a], an external SMT solver used as a black-box. The experimental evaluations in [BPV09, BPSV09], have shown that SMT can be competitive on benchmarks requiring substantial Boolean reasoning. Remarkably, the tool was able to score, at the MINIZINC Challenge competition, a gold and a silver medal in 2010 and two silver medals both in 2011 and 2012.

Comment. Part of the work presented in this dissertation falls in the same track as that of [BPV09, BPSV09, BSV10, BPSV12]. As illustrated in Section §5.3.2, we extended OPTIMATHSAT with a a new interface for dealing with MINIZINC and FLATZINC models, that can be used to solve CSP problems directly as well as to convert the input model in the *extended* SMT-LIBV2 *format* used by OMT solvers.

Differently than in the work of Bofill et al., the main optimization procedure in OPTIMATHSAT is *inlined* with the underlying SMT solver, an approach that has been shown to be more efficient than using the SMT solver as a black-box [ST12]. A second, important, difference is that the framework presented in [BSV10, BPSV12] targets the Version 1 of the SMT-LIB, and does not support any of the optimization extensions to the SMT-LIBV2 standard used by OMT solvers. The fact that the original framework is closed source, with only the binaries being freely distributed, and seemingly no longer maintained, also made it necessary to provide a new alternative to prosecute the research on this track. Last, we notice that the existing tools produce SMT-LIBV2 formulas in which neither the original Boolean structure nor global constraints are retained, making OMT solvers potentially less efficient at handling these problems. The MINIZINC interface implemented in OPTIMATHSAT makes an effort to overcome this limitation, wherever possible.

In a different set of studies, various authors considered the problem of directly encoding CSP instances in the input format used by SMT solvers, and compared various alternative

formulations of the same CSP problem to identify the modeling approach yielding the best performance.

In [FP14], Frisch et al. found that even tiny changes in the encoding of a CSP instance have a significant impact on the performance of an SMT solver. Not only that, the experimental evaluation has also shown that the various SMT solvers under consideration respond differently to changes in the encoding of the CSP problem, meaning that the same encoding could work well for some SMT solvers but be very poor for others. Given the prominent role of *cardinality constraints* in CSP problems, Frisch et al. compared various such encodings in [FG10], looking for the best alternative when dealing with a SAT solver.

In a related study, [ABP+13], Ansótegui et al. have shown that SMT can be an interesting, and potentially competitive, approach for dealing with *Weighted* CSP (WCSP) instances. A *Weighted* CSP problem is obtained from an over-constrained CSP instance by relaxing some of its constraints. Similarly to *Partial Weighted* MAXSMT, described in Section §2.3.3, *Weighted* CSP admits the use of optional weights to establish a satisfaction priority among the various soft- and meta-constraints. These weights do not have to be constant-valued. The *Weighted* CSP*s* instances in [ABP+13], encoded in some extended version of the MINIZINC format [ABP+11a, ABP+11c, ABP+13], are compiled either into COP or into *Partial Weighted* MAXSMT, and then solved. The experimental evaluation included in [ABP+13], performed over benchmarks encoding the *Nurse Rostering Problem* (NRP) and the *Balanced Academic Curriculum Problem* (BACP), compared the CSP solver CPLEX, [IBM10], with the SMT solver YICES, [DdM06a].

In [AGJ+14], Akgun et al. have shown that flattening MINIZINC models into a FLATZINC instances can introduce model-based symmetries that are not contained in the original CSP formulation. We recall here that symmetries in the encoding of a problem can lead to search redundancy and thus longer solving time, and that one common approach to mitigate this problem is to introduce lexicographic ordering constraints in the formulation of the problem to break these symmetries. Therefore, in [EF14] Elgabou et al. studied the problem of finding the best encoding for lexicographic ordering constraints in terms of the efficiency of the corresponding SMT search.

Comment. On the whole, we can summarize the outcome of these studies as follows. On the one hand, SMT can be a potentially interesting and efficient technology for dealing with CSP, especially in the case of problems requiring substantial Boolean reasoning such as the scheduling instances in [ABP+11b]. On the other hand, modeling CSP problems for SMT solvers requires a higher-level of expertise because the same CSP instance can have many possible alternative formulations, but the performance of SMT solvers on each encoding are hardly predictable in advance. Having learned this lesson from the research literature, when

developing a new framework for solving CSP problems with *Optimization Modulo Theories* (see Section §5.3.2), we decided to target our implementation over a specific OMT solver, OPTIMATHSAT [opt], to take full advantage of its strengths and features.

A significant branch of the literature went in the opposite direction, and investigated how to extend the benefits of efficient SAT and SMT techniques to the domain of CSP tools and problems. For instance, Ohrimenko et al. presented a *Finite Domain Constraint Propagation* solver built on top of a SAT solver in [OSC07, OSC09]. For this purpose, the SAT engine was extended by the authors with lazy clause generation procedures mimicking the deduction step of a typical finite domain propagator. In a closely related study, [FS09], Feiydy et al. presented a FDCP solver extended with an internal SAT engine, used for recording clauses learned along the search and quick backjumping in the search.

To conclude, we mention that after the achievements of Bofill et al. at the MINIZINC Challenge with their work on FZN2SMT [BSV10, BPSV12], that earned them a total of one gold and five silver medals over the span of three years, there has been an increasing number of SAT/SMT tools participating at the annual MINIZINC Challenge. This is the case of HAIFACSP [VS10, VS15, VS16] a CSP solver using ideas from CSP and SAT literature, and PICATSAT [ZK17] that uses the LINGELING [Bie18] SAT solver as a black-box. Remarkably, the former won two gold and a silver medal in 2016 and a bronze medal in 2018, and the latter won a total of three silver medals and three bronze medals from 2016 to 2018.

MILP and LGDP. For a long time, MILP problems have been efficiently solved by a combination of LP, *branch-and-bound* search and various *cutting-plane* methods (see e.g. [Lod09]). More recently, SAT techniques have also been integrated in these decision procedures for MILP problems (see, e.g., [ABKW08]).

Linear Disjunctive Programming (LDP) problems are LP problems where linear constraints are connected by conjunctions and disjunctions [Bal98]. *Linear Generalized Disjunctive Programming (LGDP)* is a generalization of LDP that has been proposed in [RG94] as an alternative model to the MILP problem. In contrast with MILP, which is entirely based on algebraic equations and inequalities, LGDP allows for combining algebraic and logical equations with Boolean propositions through Boolean operators, that results in a much more natural representation of discrete decisions. The state-of-the-art approach for dealing with LGDP problems is through some MILP reformulation, [RG94, VG04, SG05, SG12], by means of an efficient encoding of disjunctions and logic propositions that can be handled with some efficient MILP solver such as CPLEX [IBM10]. An important point is that LGDP and OMT($\mathcal{LRA}$) can be encoded into each other. In this regard, the experimental evaluation in [ST15a] has shown

that OMT tools can be a competitive alternative to state-of-the-art MILP solvers on problems requiring a significant amount of Boolean reasoning.

Another, notable, example of this research trend is represented by [MP13], in which Manolios et al. presented *Integer Linear Programming Modulo Theories* (IMT), a sound and complete framework for combining ILP with a background solvers for signature-disjoint stably-infinite theories $\mathcal{T}$, and INEZ, a novel IMT solver. Differently than SMT, that is centered around a SAT solver, in IMT the search is guided by a Branch and Cut procedure that communicates with some $\mathcal{T}$*-solver* by means of interface difference logic inequalities that, intuitively, serve the same purpose as interface variables for theory combination in SMT (see Section §2.2.4). In their experimental evaluation, the authors have shown that IMT can be competitive with respect to SMT on a set of benchmarks derived from the problem of synthesizing architectural models for a Boeing 787 Dreamliner. We notice that the approach of [MP13] cannot combine ILP with $\mathcal{LRA} \cup \mathcal{T}$, because $\mathcal{LIA} \cup \mathcal{T}$ and $\mathcal{LRA} \cup \mathcal{T}$ are not signature-disjoint (see Definition 2 in [MP13]).

Chapter 3

Related Work

In Chapter 2, we introduced the reader to the background and state of the art of *Optimization Modulo Theories*, covering the relevant scientific literature published prior to the start of this Ph.D. (November, 2014). In this chapter, we briefly examine those scientific publications that have been published after the beginning of this Ph.D. study and that are relevant to us.

Recent Works.

Z3 [BP14, BPF15]. In [BP14, BPF15], Bjorner et al. presented νZ, an OMT solver that was initially built as an extension of Z3 [z3] and then later on merged with the underlying SMT solver[8].

For what concerns single-objective optimization, Z3 supports OMT($\mathcal{LIRA} \cup \mathcal{T}$), OMT($\mathcal{BV} \cup \mathcal{T}$), OMT($\mathcal{PB} \cup \mathcal{T}$) and MAXSMT solving.

- To the best of our knowledge, the OMT($\mathcal{LIRA} \cup \mathcal{T}$) functionality in Z3 is similar to the one described in Section §4.1, and it also benefits from specialized algorithms for unbounded-solution detection and bound-tightening.
- According to [NR16], which cites a private communication with the authors of [BP14, BPF15] as its source, Z3 deals with OMT($\mathcal{BV} \cup \mathcal{T}$) through a reduction to partial weighted MAXSMT, as described in Section §4.3.
- Z3 features several specialized engines for dealing with OMT($\mathcal{PB} \cup \mathcal{T}$) and MAXSMT problems including, among others, WMAX and MAXRES. The former approach uses a specialized theory solver of *costs* similar to [CFG$^+$10], that we describe in Section §2.3.3. The latter approach combines the core-guided *Maximum Resolution* MAXSAT *Engine*,

[8] For this reason, we henceforth refer to either tool with the name Z3.

presented by Narodytska et al. in [NB14], with the lazy-SMT solving framework. With this approach, all clauses are initially asserted as "hard", and an inference rule is used to progressively relax the initial set of clauses so that to remove any conflict —found along the search— involving some "soft" clause, up until the set of "soft" clauses becomes empty or a purely "hard" conflict is found. Since this method has also been added to OPTIMATHSAT (later on), we provide an in-depth description of MAXRES in Section §4.2.2.

To the best of our knowledge, Z3 also ships with preprocessing techniques that re-encode the 0-1 integer variables of the input formula into Pseudo-Boolean or MAXSMT constraints, for better performance. In addition, Z3 features a Pseudo-Boolean $\mathcal{T}$*-solver* that can generate sorting circuits on demand for Pseudo-Boolean inequalities featuring sums with small coefficients [BPF15, Bjo16].

When dealing with multiple objectives, Z3 supports Multiple-Independent (a.k.a Boxed), Lexicographic and Pareto optimization.

- Z3's implementation of Multiple-Independent OMT is similar to the one available in OPTIMATHSAT, that is described in Section §4.6.2. The main difference among the two implementations is that in Z3 the optimization functionality is built on top of the underlying SMT solver, similarly to the offline approach described in Section §2.3.1. In addition, Z3 uses ad hoc techniques for detecting unbounded solutions.
- To the best of our knowledge, Z3's implementation of its Lexicographic optimization algorithm is not documented in any publication. We describe OPTIMATHSAT's implementation of Lexicographic optimization in Section §4.6.3.
- The Pareto optimization approach implemented by Z3 is based on the *Guided Improvement Algorithm* presented in [REJ09]. In Section §4.6.4 we describe a very similar approach, implemented in OPTIMATHSAT, and also a different Pareto optimization algorithm based on lexicographic optimization.

Similarly to OPTIMATHSAT, in Z3 optimization is supported with any combination of theories. Moreover, both OMT solvers are incremental, and allow for pushing and popping both objectives and clauses on the internal stack of formulas. In Section §4.5, we describe how this feature is achieved in OPTIMATHSAT.

HAZEL [NR16]. Another, relevant, related work is [NR16]. In this paper, Nadel et al. originally presented the *Bit-Vector Optimization with Weak Assumptions* (OBV-WA) and the *Bit-*

Vector Optimization with Binary Search (OBV-BS) algorithms, both of which can deal with *unsigned* OMT($\mathcal{BV} \cup \mathcal{T}$) formulas quite efficiently.

The OBV-WA algorithm modifies the decision and backtrack mechanism of the underlying SMT solver, transforming the bits of the $\mathcal{BV}$ goal into high-priority decision variables with preset phase-saving values, to force the optimization search to explore the search-space starting from the more-than-optimal (unsatisfiable) subregion.

The OBV-BS algorithm performs a binary search exploration over the bits of the objective function, using a sequence of incremental calls to the underlying SMT solver and, optionally, phase-saving initialization for the better performance.

Both algorithms are fully incremental, and have been implemented within Intel's eager BV solver HAZEL [Nad14].

In this dissertation, we describe a generalization of these two algorithms to the case of both *signed* and *unsigned* $\mathcal{BV}$ optimization, that we have implemented in OPTIMATHSAT after learning about [NR16]. OPTIMATHSAT's implementations of the OBV-WA and OBV-BS algorithms are described in Sections §4.3.2 and §4.3.3 respectively.

MAXHS-MSAT [FBB18]. Quite recently, Fazekas et al. presented in [FBB18] a new framework for dealing with MAXSMT based on the state-of-the-art Implicit Hitting Set (IHS) algorithm. The latter is a MAXSAT algorithm that combines an Integer Programming (IP) solver, used to generate candidate sets of soft clauses that *hit* a set of constraints in an optimal way, with a SAT solver for checking satisfiability. In their paper, the authors have described a general formal reasoning calculus for lifting the IHS method to the SMT level, to deal with MAXSMT. The paper describes an experimental evaluation, including a comparison with OPTIMATHSAT and Z3, showing the benefits of this approach.

PULI [KBE18]. In another recent work, Kovásznai et al. presented PULI, a novel OMT solver for quantifier free formulas with Uninterpreted Functions (UF) and Linear Integer Arithmetic (LIA). Differently from other OMT solvers, PULI applies linear regression analysis over a user-defined resource function to speed up the convergence of the OMT solver towards the optimal solution. An even more significant performance improvement is obtained when dealing with monotonous OMT problems. In their paper, the authors included an experimental evaluation performed over sets of benchmarks derived from their *Wireless Sensor Networks* (WSNs) OMT application described in [KBE17, KEB18] (see Section §7.1) and also on the well-known Knapsack problem. The experiments show the validity of this approach and its effectiveness in speeding up the basic OMT procedures described in Section §2.3.1.

CEGIO [ABCF16, AAdB$^+$17, AAdB$^+$18]. Recently, Araujo et al. presented a *Counterexample Guided Inductive Optimization* (CEGIO) algorithm based on *Satisfiability Modulo Theories*, for dealing with the optimization of a wide-range of functions, including non-linear and non-convex problems using fixed-point arithmetic. In contrast with the *Optimization Modulo Theories* described in this dissertation, in [ABCF16, AAdB$^+$17, AAdB$^+$18] the optimization search is iteratively advanced by solving a sequence of SMT formulas and by analyzing their corresponding counterexamples generated with an internal SMT solver used as a black-box. Each SMT formula consists in a verification problem automatically generated with Bounded Model Checking (BMC) techniques and derived from an ANSI-C model of the original optimization problem. Similarly to OMT, the CEGIO optimization algorithm guarantees the optimality of the solution even when other, traditional, techniques can get trapped by local minima. Experimental results included in [ABCF16, AAdB$^+$17, AAdB$^+$18] show the benefits of this approach.

Other Works.

Some OMT solvers appeared prior to the start of this Ph.D. study. Therefore, we have already described their techniques in Chapter 2. For clarity of illustration and an easier comparison with OPTIMATHSAT, we briefly recap some important details about these OMT solvers, that would otherwise remain scattered throughout this dissertation.

BCLT [NO06, BNO$^+$08, Roc11, LORR14]. With their pioneering work in [NO06], Nieuwenhuis and Oliveras presented BCLT, the first SMT solver with optimization capabilities that has appeared on the scene of Satisfiability Modulo Theories, and paved the way for the advent of Optimization Modulo Theories. Remarkably, BCLT has also been the first OMT solver featuring optimization procedures for the general case of *Integer Linear Arithmetic* optimization [Roc11]. These procedures have been used in [LORR14] to extend BCLT with a decision procedure for polynomial constraints, that is, *Non-Linear Integer Arithmetic*.

In this dissertation, we describe the OMT($\mathcal{LIA} \cup \mathcal{T}$) procedures of BCLT in Section §2.3.2, and illustrate the OMT($\mathcal{PB} \cup \mathcal{T}$) and MAXSMT handling of BCLT in Section §2.3.3. The optimization procedures described in [NO06, BNO$^+$08, Roc11], are not incremental and, for what concerns OMT($\mathcal{LIA} \cup \mathcal{T}$), do not allow for mixed integer/real optimization.

SYMBA [LAK$^+$14]. SYMBA is an OMT($\mathcal{LRA} \cup \mathcal{T}$) solver built on top of the Z3 SMT solver used as a black-box, that allows for optimizing multiple $\mathcal{LRA}$ objectives according to the Multiple-Independent combination approach. For a detailed description of the OMT($\mathcal{LRA} \cup \mathcal{T}$) procedure of [LAK$^+$14], we refer the reader to the last part of Section §2.3.1. In [LAK$^+$14],

the authors note that the OMT solver is not incremental, it does not handle strict $\mathcal{LRA}$ inequalities (i.e. $>$, $<$) and it only supports combination of theories as long as the theory $\mathcal{T}$ is signature-disjoint with $\mathcal{LRA}^{\leq}$. OMT($\mathcal{LIRA} \cup \mathcal{T}$), in particular, is not supported.

We recall that OMT($\mathcal{PB} \cup \mathcal{T}$) and MAXSMT can be encoded into each other, and that both are strictly less general than OMT($\mathcal{LRA} \cup \mathcal{T}$). In Section §2.3.3, we described various techniques for dealing with this kind of problems, including, e.g., [NO06, CFG+10, ABP+11b, CGSS13a].

Here, we also mention that the SMT solver YICES [DM06b] provides support for MAXSMT, although there is no publicly-available documentation of its procedures. In addition, Ansótegui et al. [ABP+11b] describe the evaluation of an implementation of a MaxSMT procedure based on YICES, although this implementation is not publicly available.

Part II

Contributions

Chapter 4

Advances in OMT

This chapter illustrates major advances in the context of Optimization Modulo Theories that occurred during this Ph.D. study.

This chapter is organized as follows:

§4.1 *OMT ($\mathcal{LIRA} \cup \mathcal{T}$)*: OMT with *Mixed Linear Integer Arithmetic* cost functions [ST15c].

§4.2 *OMT ($\mathcal{PB} \cup \mathcal{T}$)*/MAXSMT: OMT with *Pseudo-Boolean* and MAXSMT cost functions. In §4.2.1, we consider the standard OMT-based search enriched with sorting networks [ST17], whereas in §4.2.2 we illustrate an alternative approach based on the MAXRES engine [NB14, BP14].

§4.3 *OMT ($\mathcal{BV} \cup \mathcal{T}$)*: OMT with *Bit-Vector* cost functions. In §4.3.1, we illustrate the standard OMT-based search, whereas in §4.3.2 and in §4.3.3 we describe the *Bit-Vector Optimization with Weak Assumptions* (OBV-WA) and the *Bit-Vector Optimization with Binary Search* (OBV-BS) algorithms respectively, both of which were originally presented in [NR16].

§4.4 *OMT ($\mathcal{FP} \cup \mathcal{T}$)*: OMT with *Floating-Point* cost functions [TS19]. We consider a standard OMT-based approach first (§4.4.1), and then present the novel *Floating-Point Optimization with Binary Search* (OFP-BS) algorithm in §4.4.2.

§4.5 *Incremental OMT*: a description of a two useful techniques for creating an *incremental* Optimization Modulo Theories solver [ST15c].

§4.6 *Multi-Objective Optimization*: a definition of the problem, followed by a detailed analysis of a variety of multi-objective combinations that are supported by OMT solvers [LAK+14, BP14, BPF15, ST15b, ST15c]. We include in our presentation a description of: MINMAX/MAXMIN *Combination* (§4.6.1), *Multiple-Independent Optimization* (§4.6.2), *Lexicographic Optimization* (§4.6.3) and *Pareto Optimization* (§4.6.4).

§4.7 *All-OMT*: a simple extension of *All-SMT* to the case of Optimization Modulo Theories.

Full Disclosure. Most of the material presented in the following sections is taken from publications co-authored by the Ph.D. candidate in collaboration with Prof. Roberto Sebastiani[a] [ST15b, ST15c, ST17, ST18, TS19].

In order to maintain the flow of the discourse, and as a premise to the description of OPTIMATHSAT in Chapter §5, we also include innovations not co-authored by the Ph.D. candidate but implemented in OPTIMATHSAT (like, e.g., [NB14, BP14, NR16]). When this is the case, we include in the incipit of the corresponding section an explicit full disclosure statement clearly stating the original authors of the methods being described and referencing the source material.

We note that, in this dissertation, the description of these methods is centered around OPTIMATHSAT's implementation. When this differs from the one described in the original work, we highlight these differences and refer to the original publication for more details on the original approach.

[a]Prof. Roberto Sebastiani, roberto.sebastiani@unitn.it, DISI, University of Trento, Italy.

4.1 OMT ($\mathcal{LIRA} \cup \mathcal{T}$)

In the following, we describe the OMT($\mathcal{LIRA} \cup \mathcal{T}$) handling of OPTIMATHSAT presented in [ST15c], which is based on the OMT($\mathcal{LIA} \cup \mathcal{T}$) extension of OPTIMATHSAT presented in [Tre14].

The *Optimization Modulo Theories problem* for *Linear Integer and Rational Arithmetic* cost functions is defined as follows.

Definition 4.1.1. *(OMT($\mathcal{LIRA} \cup \mathcal{T}$), OMT($\mathcal{LIRA}$)). Let φ be a ground SMT($\mathcal{LIRA} \cup \mathcal{T}$) formula and* obj *be a $\mathcal{LIRA}$ variable occurring in φ. We call an Optimization Modulo $\mathcal{LIRA} \cup \mathcal{T}$ problem, the problem of finding a model $\mathcal{M}$ for φ (if any) whose value of* obj*, denoted with* $\mathsf{min}_{\mathsf{obj}}(\varphi)$*, is minimum. We call an Optimization Modulo $\mathcal{LIRA}$ problem, written OMT($\mathcal{LIRA}$), an OMT($\mathcal{LIRA} \cup \mathcal{T}$) problem where $\mathcal{T}$ is the empty theory. (The dual definition where we look for the maximum follows straightforwardly)*

We note that OMT($\mathcal{LIRA} \cup \mathcal{T}$) is conceptually different from both OMT($\mathcal{LRA} \cup \mathcal{T}$) (described in §2.3.1) and OMT($\mathcal{LIA} \cup \mathcal{T}$) (described in §2.3.2), as it allows for a mixed use of Integer and Rational variables and constraints.

Remark 4.1.1. In general, the solution of a *bounded* OMT($\mathcal{LIA} \cup \mathcal{T}$) problem can be found through a simple enumeration of all possible models $\mathcal{M}$ of φ, since in this case there are only finitely many of them. This can be done, for example, using either the *linear-* or the *binary-search* schemata of [ST12, ST15a] described in §2.3.1. However, this approach is insufficient both when the objective function obj is not lower-bounded and when obj is a mixed $\mathcal{LIRA}$ expression. In these cases, a $\mathcal{LIA}$-minimization procedure is necessary to guarantee termination. In addition, the availability of such a procedure can speed up the optimization search by preventing a truth assignment μ from being generated multiple times along the search.

In OPTIMATHSAT, the *inline schema* for $\mathcal{LRA}$ optimization described in Section §2.3.1 is adapted to deal with the case of $\mathcal{LIRA}$ objectives by replacing the procedure $\mathcal{LRA}$-MINIMIZE with a novel $\mathcal{LIRA}$-MINIMIZE one. Given a complete truth assignment μ, such that μ propositionally satisfies φ, the function $\mathcal{LIRA}$-MINIMIZE performs an initial unboundedness test and then, if obj is bounded in correspondence with μ, searches the optimal value of obj by leveraging the $\mathcal{LIRA}$-solver implemented in MATHSAT5 [Gri12, CGSS13b].

Unboundedness test. Given an OMT($\mathcal{LIRA} \cup \mathcal{T}$) problem $\langle \varphi, \mathsf{obj} \rangle$, an easy-to-see property is that obj can only be unbounded over φ if the corresponding OMT($\mathcal{LRA} \cup \mathcal{T}$) instance $\langle \varphi', \mathsf{obj}' \rangle$, obtained by discarding the integrality constraints on any Integer variable in the original problem, is unbounded [BGH87]. Therefore, given a truth assignment μ and the corresponding set of $\mathcal{LIRA}$ constraints, we can check whether obj is $\mathcal{LIRA}$-unbounded for μ with a simple run of the $\mathcal{LRA}$-MINIMIZE() procedure of §2.3.1. Invoking the latter procedure has three possible outcomes. The first is that obj is found to be unbounded, and therefore $-\infty$ can be returned. The second possible outcome is that obj is bounded and the resulting optimum model $\mathcal{M}$ happens to not violate any integrality constraint of the original problem. In this case, both the model $\mathcal{M}$ and the corresponding value of obj can be returned with no further effort. The third case is when the $\mathcal{LRA}$-optimal model $\mathcal{M}$ violates (some of) the integrality constraints in the original problem. In this case, $\mathcal{LIRA}$-MINIMIZE() performs a local Branch&Bound search to assign each Integer variable an integral value.

Branch&Bound. In the literature, Branch&Bound is a well-known approach for exploring the feasible space of a *Mixed Integer Linear Programming* (MILP) problem. Roughly speaking, the problem of finding an optimal integral solution is reduced to solving a sequence of *Linear Programming* (LP) subproblems in which an increasing number of Integer variables is forced to evaluate to an Integer value with the use of additional linear constraints blocking any undesired

assignment of values.

In OPTIMATHSAT, the underlying SMT solver MATHSAT5, [CGSS13b], is already provided with a built-in Branch&Bound procedure for checking $\mathcal{LIRA}$-satisfiability [Gri12]. This is adapted for dealing with the optimization of a $\mathcal{LIRA}$ goal obj as follows. First, the function $\mathcal{LRA}$-MINIMIZE() of §2.3.1 is invoked on each node of the Branch&Bound tree, to ensure that the value of obj is always locally optimal with respect to the $\mathcal{LRA}$-domain associated with the node itself. Second, each time a new integral model $\mathcal{M}$ is found in correspondence with some node of the Branch&Bound tree, the Branch&Bound search is restarted and a new linear cut of the form $(\mathsf{obj} < \mathsf{ub})$ is pushed on the local stack of constraints. Since the value of obj is known to be bounded, it follows that the optimal value of obj must be necessarily found after a finite number of restarts, after which the local stack of constraints becomes unsatisfiable. At this point, the most recently found integral model $\mathcal{M}$ and the corresponding $\mathcal{LIRA}$-optimal value of obj can be returned.

This particular restart-based approach to $\mathcal{LIRA}$ optimization is enabled by the implementation of the Branch&Bound decision procedure for $\mathcal{LIRA}$ in MATHSAT5, that features several advanced features designed for search efficiency. Two features are particularly noteworthy. The first is the use of historical information to drive the Branch&Bound search across subsequent runs, so that good past decisions are replicated whenever possible. The second is an internal *backjumping* mechanism based on conflict-set analysis, that allows the OMT solver to automatically discard a large number of unsatisfiable LP subproblems without exploring them one by one. For more details about these and other enhancements, we refer the reader to [Gri12].

Improvements. In most situations, the above Branch&Bound implementation is reasonably efficient in practice, so much so that the optimal value of obj is found with very few restarts, often even in the first run. However, as widely known in the literature, there are some degenerate cases in which the Branch&Bound approach becomes very inefficient and has some difficulty in finding the optimal solution. In these situations, other techniques such as cutting planes can be employed to rescue the OMT solver. Since such advanced techniques are already made available by the MATHSAT5 $\mathcal{LIRA}$-solver, we implemented a *truncated* variant of the Branch&Bound search in which $\mathcal{LIRA}$-MINIMIZE() stops as soon as it finds its first integral model or it exhausts its budget. Then, the procedure returns the suboptimal value ub of obj to the CDCL search loop, that learns a constraint of the form $(\mathsf{obj} < \mathsf{ub})$ as in §2.3.1. On the one hand, this has the advantage that the next satisfiability check involving the $\mathcal{LIRA}$-solver will now use the entire stack of specialized routines it contains, including cutting planes, and not just the Branch&Bound module. If the stack of formulas is still satisfiable, this guarantees a relatively cheap improvement of the cost function value. On the other hand, this approach

might cause the OMT solver to consider a truth assignment μ more than once.

Experimental Evaluation. In Section §6.1, we compare the implementation of these techniques on top of OPTIMATHSAT with other state-of-the-art OMT solvers.

4.2 OMT ($\mathcal{PB} \cup \mathcal{T}$)/MAXSMT

Both OMT($\mathcal{PB} \cup \mathcal{T}$) and MAXSMT constitute two important —and frequent— subcases of OMT($\mathcal{LRA} \cup \mathcal{T}$). To this aim, in the *Background & State of the Art*, Section §2.3.3, we illustrated a variegated number of techniques for dealing with these two problems that have been previously proposed in the literature.

We describe two additional techniques that have been recently proposed in the context of *Optimization Modulo Theories*. The first approach, is a revisited version of the OMT($\mathcal{LRA} \cup \mathcal{T}$) encoding (see §2.3.3) for OMT($\mathcal{PB} \cup \mathcal{T}$) and MAXSMT that exploits sorting networks to gain a significant performance advantage. This method, which we describe in full detail in Section §4.2.1, was first presented in [ST17]. The second approach is MAXRES, a core-based MAXSAT engine first presented by Narodytska et al. in [NB14] and then used to deal with MAXSMT problems by Bjorner et al. in [BP14]. We illustrate MAXRES in Section §4.2.2.

4.2.1 Sorting Networks Approach

As described in Section §2.3.3, an option for dealing with OMT($\mathcal{PB} \cup \mathcal{T}$) and MAXSMT is to encode these problems into an OMT($\mathcal{LRA} \cup \mathcal{T}$) pair $\langle \varphi, \mathsf{obj} \rangle$, so that the inline OMT optimization procedures described in Section §2.3.1 can then be used to find the optimal solution. However, in [ST17] we have shown that a naive application of this approach can suffer from poor performance with some types of formulas and so it would benefit from the use of more sophisticated techniques such as Sorting Networks.

In this section, we provide a detailed description of the performance issues and the solution, based on Sorting Networks, proposed in [ST17]. We also note that, henceforth, we will focus our description on the case of $\mathcal{PB}$ objectives only to simplify the discussion. This does not cause any loss of generality, since any MAXSMT goal can be rewritten as a OMT($\mathcal{PB} \cup \mathcal{T}$) problem, as shown in Section §2.3.3.

Performance issues [ST17].

In [ST17], we observed that the usual optimization with linear-search (Section §2.3.1) can end up generating exponentially many Theory Lemmas when dealing with a $\mathcal{PB}$ objective obj in

which all weights have the same value w:

$$\mathsf{obj} \stackrel{\text{def}}{=} w \cdot \sum_{i=0}^{n-1} A_i. \tag{4.1}$$

Let obj be an objective as in Equation (4.1) to be minimized, and μ be a satisfiable (and total) truth assignment found by the OMT solver during the search. Then, given $A_T = \{A_i \,|\, \mu \models A_i\}$ and $k = |A_T|$, the upper bound value of obj in μ is $\mathsf{ub} = w \cdot k$. As described in Section §2.3.1, the OMT solver learns a unit clause in the form $\neg(\mathsf{ub} \leq \mathsf{obj})$ for each truth assignment μ found in linear-search mode. Learning this unit clause removes the current truth assignment μ from the feasible search space, which is thus narrowed as a result, and forces the OMT solver to search for another truth assignment μ' —with a smaller upper bound ub'— in the next iteration of the optimization search.

As observed in [ST17], the effect of learning the unit clause $\neg(\mathsf{ub} \leq \mathsf{obj})$ is not limited to the removal of the current truth assignment μ from the feasible space. In fact, it also makes inconsistent any other (partial) truth assignment μ' setting exactly k (or more) variables A_i to True. More precisely, learning such clause prunes at least $\gamma = \binom{n}{k}$ truth assignments from the search space, where γ is the number of of possible permutations of μ over the variables A_i.

Regrettably, since the unit clause $\neg(\mathsf{ub} \leq cost)$ is a $\mathcal{LRA}$ term, the CDCL engine is unable to determine by simple Boolean Constraint Propagation (BCP) the resulting inconsistency of any (partial) truth assignment μ' setting exactly k variables to True. It is not until the *$\mathcal{T}$-solver* for linear rational arithmetic is invoked that such inconsistency can be revealed, and a conflict clause blocking the (partial) truth assignment μ' can be learned as a Theory Lemma. This reliance on the $\mathcal{LRA}$-Solver to reveal such inconsistencies presents two major drawbacks. The first is that the *$\mathcal{T}$-solver* for linear rational arithmetic is much more resource-demanding than BCP. The second issue is that both SMT and OMT solvers invoke the $\mathcal{LRA}$-Solver less frequently than BCP, precisely to amortize its cost. As a result, the OMT solver can perform poorly when dealing with this kind of objectives.

Note that the exact same problematic arises when the OMT solver learns a unit clause of the form $\neg(\mathsf{pivot} \leq \mathsf{obj})$ in a binary-search step during the minimization of obj. When obj is maximized, a dual case occurs.

Example 4.2.1. *Figure 4.1 depicts a toy example of OMT search over the pair $\langle \varphi, \mathsf{obj} \rangle$, where φ is an SMT formula and* $\mathsf{obj} \stackrel{def}{=} \sum_{i=1}^{4} A_i$ *(i.e., $w_i = 1$ for every i).*

Since obj *is a Pseudo-Boolean objective, we assume that the problem has been first encoded into OMT($\mathcal{LRA} \cup \mathcal{T}$) using Equations* (2.2a)-(2.2c)*, so that (1) the input formula φ is now extended with $\cup_{i=1}^{4}\{(\neg A_i \vee (x_i = 1)), (A_i \vee (x_i = 0)), (0 \leq x_i), (x_i \leq 1)\}$ and (2) the goal* obj *is rewritten as $\sum_{i=1}^{4} x_i$.*

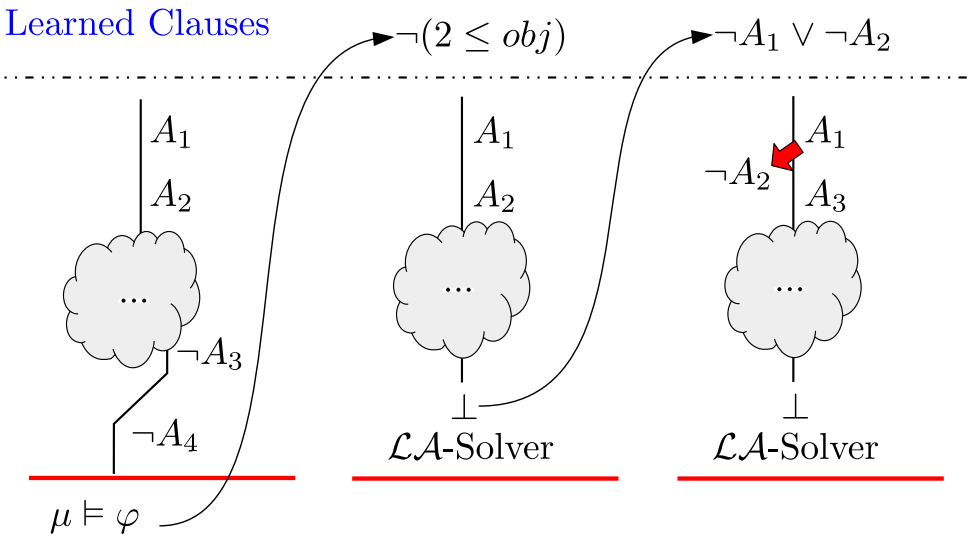


Figure 4.1: A simple example of OMT search with a $\mathcal{PB}$ objective (Figure taken from [ST17]).

Let's suppose that, as depicted on the left-hand side of Figure 4.1, the OMT solver finds a truth assignment μ such that $\mu \models \varphi$ and $\{A_1, A_2, \neg A_3, \neg A_4\} \subseteq \mu$. Then, the value of the objective function in μ, i.e. 2, is a new upper bound for obj*. Thus, the unit clause $\neg(2 \leq \mathsf{obj})$ is learned and the Boolean search is restarted to find a better solution (if any).*

In the subsequent run of the Boolean search, depicted at the center of Figure 4.1, A_1 and A_2 are again decided[9]*. As a consequence, the (partial) truth assignment μ' that is now being constructed contains $\{\neg(2 \leq \mathsf{obj}), (x_1 = 1), (x_2 = 1)\}$, which is $\mathcal{LRA}$-inconsistent. However, this inconsistency in the (partial) truth assignment μ' is not revealed up until when the (more expensive) $\mathcal{LRA}$-*Solver *is invoked, which, depending on the early-pruning strategy implemented in the OMT solver, can happen after μ' has been further extended by BCP.*

*Eventually, the $\mathcal{LRA}$-*Solver *determines the inconsistency of μ', and the OMT solver learns the conflict clause $\neg A_1 \vee \neg A_2$ to prevent the conflicting assignment from being generated again. The search is then forced to backjump and toggle the value of A_2, as shown on the right-hand side of Figure 4.1. At this point, a completely legitimate scenario is that A_3 is decided to be true, causing a new conflict with respect to the unit clause learned by the optimization search, and so on. In this way, the solver can pointlessly enumerate and check all the up-to $\binom{4}{2}$ truth assignments that are (1) consistent with φ and that (1) assign two variables A_i to True at the same time, despite of the fact that none of these truth assignments is compatible with $\neg(2 \leq \mathsf{obj})$.* ◇

[9] We note that this is not a remote possibility, due to the effect of the phase-saving technique that is applied in both SMT and OMT solvers.

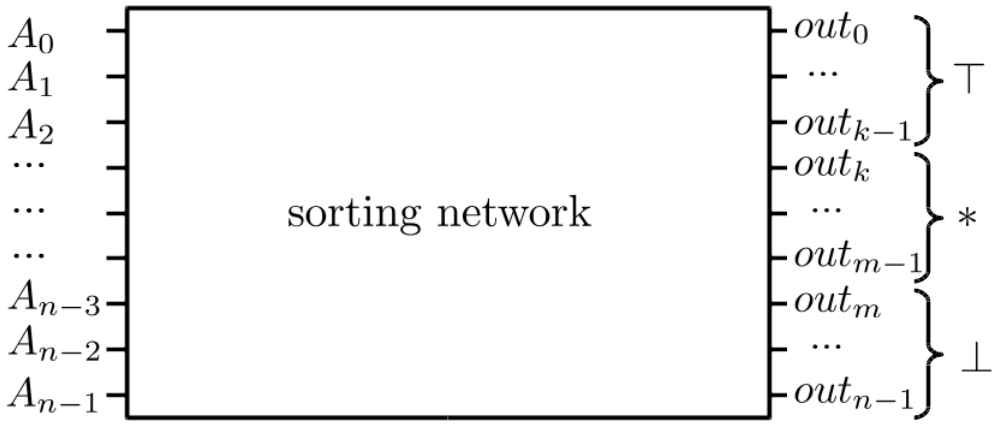


Figure 4.2: The basic schema of a sorting network relation (Figure taken from [ST17]).

The general case. The performance issue identified in [ST17] is not limited to the simple case depicted in Equation (4.1) and can be generalized to any $\mathcal{PB}$ objective obj in which there are groups of Boolean variables A_i sharing the same weight:

$$\mathsf{obj} = \ \tau_1 + ... + \tau_m, \tag{4.2}$$

$$\textstyle\bigwedge_{j=1}^{m} \big((\tau_j = w_j \cdot \sum_{i=1}^{k_j} A_{ji}) \ \wedge \ (0 \leq \tau_j) \wedge (\tau_j \leq w_j \cdot k_j) \big), \tag{4.3}$$

where the logically-redundant constraints $(0 \leq \tau_j) \wedge (\tau_j \leq w_j \cdot k_j)$ are added for the same reason as with (2.2b).

OMT with Sorting Networks [ST17].

A solution for this efficiency issue, that we presented in [ST17], is to leverage *bidirectional sorting networks* so that the inconsistency of a (partial) truth assignment μ with respect to a unit clause in the form $\neg(\mathsf{ub} \leq cost)$ can be revealed earlier in the search by means of Boolean Constraint Propagation (BCP).

Definition 4.2.1. *(*Sorting network, bidirectional sorting network *[ST18]). We call* sorting network *a relation among n input Boolean variables A_i and n output Boolean variables out_i, as illustrated in Figure 4.2, such that if in the current (partial) truth assignment μ, k variables are set to True, $n - m$ variables are set to False and $m - k$ are unassigned, then by Boolean Constraint Propagation $out_0, ..., out_{k-1}$ are set to True, $out_m, ..., out_{n-1}$ are set to False and $out_k, ..., out_{m-1}$ are not propagated.*

A sorting network *is said to be* bidirectional *if it is also the case that if out_{k-1} is forced to be True (that is, at least k inputs must be True) and $n - k$ inputs A_i are False, then by*

Boolean Propagation all other unassigned A_is are automatically set to True; vice-versa, if out_{k+1} is forced to be False (that is, at most k inputs can be True) and $n-k$ inputs A_i are True, then all other unassigned A_is are automatically set to False.

Given a bidirectional sorting network circuit C with n inputs $A_0, ..., A_{n-1}$ and n outputs $out_0, ..., out_{n-1}$, we encode a OMT($\mathcal{PB} \cup \mathcal{T}$) problem $\langle \varphi, \mathsf{obj} \rangle$, such that $\mathsf{obj} \stackrel{\text{def}}{=} w \cdot \sum_{i=0}^{n-1} A_i$, into an OMT($\mathcal{LRA}$) instance $\langle \varphi'', \mathsf{obj}'' \rangle$ as follows:

$$\varphi'' \stackrel{\text{def}}{=} \varphi' \wedge C \wedge \bigwedge_{k=0}^{k=n-1} \begin{cases} out_k \rightarrow ((k+1) \cdot w \leq \mathsf{obj}) \\ \neg out_k \rightarrow (\mathsf{obj} \leq k \cdot w) \\ \neg((k+1) \cdot w \leq \mathsf{obj}) \vee \neg(\mathsf{obj} \leq k \cdot w) \end{cases} \tag{4.4a}$$

$$\mathsf{obj}'' \stackrel{\text{def}}{=} \mathsf{obj}' \tag{4.4b}$$

where φ' and obj' are defined as in Equations (2.2a)-(2.2c). The logically redundant constraints at the third line of Equation (4.4a) are added to the formula so that the negation of $(\mathsf{obj} \leq (i-1) \cdot w)$ is directly implied by BCP from $(i \cdot w \leq \mathsf{obj})$ (and vice versa), without the aid of the $\mathcal{LRA}$-Solver.

The main advantage of encoding an OMT($\mathcal{PB} \cup \mathcal{T}$) problem as in Equations (4.4a)-(4.4b) is that, whenever the OMT solver (1) learns a unit clause in the form $(\mathsf{obj} < k \cdot w)$ (i.e. as a consequence of finding a satisfiable truth assignment μ in which k variables A_i are true, or as part of a binary-search step in which $\mathsf{pivot} = k \cdot w$) and (2) the search is restarted from level zero to find a new satisfiable truth assignment μ', as soon as $k-1$ inputs A_i are assigned to True in μ' then by BCP the remaining $n-k+1$ inputs are immediately propagated to False. Dually, when the OMT solver learns a unit clause in the form $(\mathsf{obj} > k \cdot w)$, as soon as $n-k$ inputs A_i are assigned to False in the (partial) truth assignment μ' under construction then by BCP the remaining k inputs are immediately propagated to True.

Example 4.2.2. *Figure 4.3 considers the same OMT problem $\langle \varphi, \mathsf{obj} \rangle$, with $\mathsf{obj} \stackrel{def}{=} \sum_{i=1}^{4} A_i$, as in Example 4.2.1. This time, however, the OMT($\mathcal{PB} \cup \mathcal{T}$) instance is encoded with Equations* (4.4a)-(4.4b) *into OMT($\mathcal{LRA} \cup \mathcal{T}$) with sorting network.*

In particular, φ is now extended with (1) the same clauses as in Example 4.2.1 (2) a bidirectional sorting-network relation C having $\{A_1, A_2, A_3, A_4\}$ as inputs and $\{out_1, out_2, out_3, out_4\}$ as outputs (3) the constraints $\bigwedge_{i=1}^{4}(out_i \rightarrow (i \leq \mathsf{obj}))$, $\bigwedge_{i=1}^{4}(\neg out_i \rightarrow (\mathsf{obj} \leq (i-1))$ and $\bigwedge_{i=1}^{4}(\neg(i \leq \mathsf{obj}) \vee \neg(\mathsf{obj} \leq (i-1)))$. The goal obj is rewritten as $\sum_{i=1}^{4} x_i$, same as before.

Let's assume that, as shown on the left-hand side of Figure 4.3, the search behaves as in Example 4.2.1 up until it finds the same satisfiable truth assignment μ such that $\{A_1, A_2, \neg A_3, \neg A_4\} \subseteq \mu$, for which the value of obj is equal to 2. Then, same as before, the OMT solver learns the unit clause $\neg(2 \leq \mathsf{obj})$ to find an improving solution, and the Boolean search is restarted.

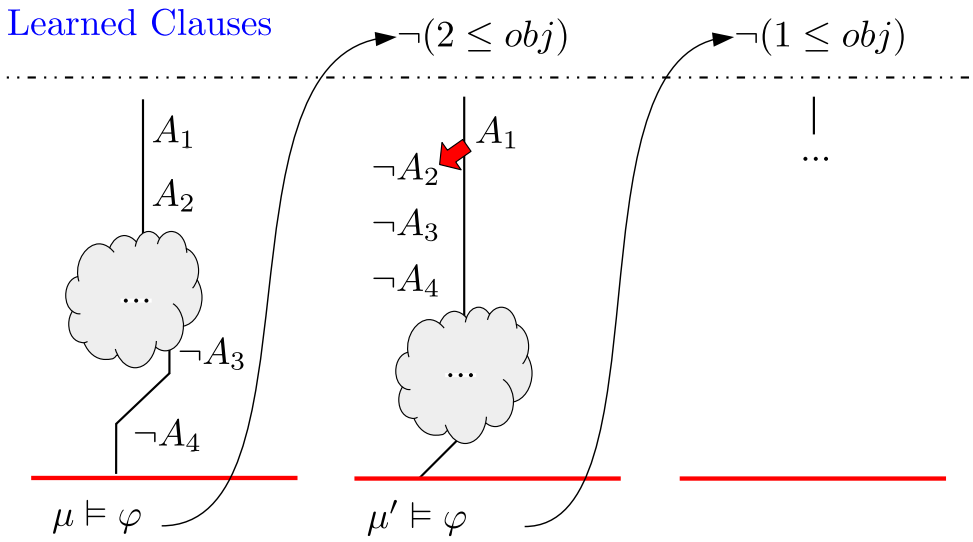


Figure 4.3: A simple example of OMT search with sorting networks (Figure taken from [ST17]).

This time, however, the search follows a different path. In fact, due to the presence on the formula stack of the bidirectional sorting network C and the constraint $out_2 \rightarrow (2 \leq \mathsf{obj})$, learning the unit clause $\neg(2 \leq \mathsf{obj})$ causes the outputs $\{out_2, out_3, out_4\}$ to be immediately unit propagated to False by BCP. Moreover, as depicted at the center of Figure 4.3, as soon as A_1 is decided then all the remaining inputs $\{A_2, A_3, A_4\}$ are immediately unit propagated to False by BCP. Compared to the scenario depicted in Example 4.2.1, this saves up to $\binom{4}{2}$ (expensive) calls to the $\mathcal{LRA}$-Solver.

When the set $\{A_1, \neg A_2, \neg A_3, \neg A_4\}$ is eventually extended to a complete truth assignment μ' such that $\mu' \models \varphi$, the OMT solver learns the new unit clause $\neg(1 \leq \mathsf{obj})$ and the search proceeds as shown on the right-hand side of Figure 4.3. ◇

Method generalization. This approach based on sorting networks can be generalized to deal with OMT($\mathcal{PB} \cup \mathcal{T}$) problems $\langle \varphi, \mathsf{obj} \rangle$ in which groups of terms share the same weight w_j, as for the objective function defined in Equations (4.2)-(4.3).

In this case, a separate sorting network circuit has to be generated for each term τ_j in (4.2)-(4.3). Then, as described in [ST17], the following constraints are introduced in the formula to ensure that the circuit is activated by BCP

$$\bigwedge_{j=1}^{m} \bigwedge_{i=1}^{k_j} (\neg(w_j \cdot i \leq \mathsf{obj}) \rightarrow \neg(w_j \cdot i \leq \tau_j)), \tag{4.5}$$

Note that these constraints are not always sufficient to avoid some (expensive) calls to the

$\mathcal{LRA}$-Solver, and generally tend do be more effective if the number of groups with uniform weight in the $\mathcal{PB}$ sum is limited. To this aim, consider the extreme case of a $\mathcal{PB}$ objective in which every Boolean variable A_i has a unique weight w_i as an example. If μ is the most recent satisfiable truth assignment and it contains A_k such that $\forall i.w_k > w_i$ and at least one other A_i, then when $\neg(\mathsf{ub} \leq cost)$ is learned none of the implications in (4.5) gets activated by BCP.

To overcome this limitation, the following workaround can be used. Each time a satisfiable μ is found, the OMT solver does not only learn $\neg(\mathsf{ub} \leq cost)$, but also the following clause

$$\bigvee_{j=1}^{m} \neg(\mathsf{ub}_{\tau_j} \leq \tau_j) \tag{4.6}$$

where ub_{τ_j} is the value of τ_j in the current truth assignment μ. This constraint forces the Boolean search to improve the value of at least one τ_j, but *possibly many more*, over the corresponding sorting circuit C_j at each linear-step of the optimization search.

Bidirectional Sorting Networks.

In OPTIMATHSAT, we have so far considered two sorting network encodings: the sequential counter encoding in [Sin05], that we have extended to be bidirectional, and the cardinality network encoding in [ANORC11, ANOR13].

Bidirectional Sequential Counter Encoding [Sin05]. In [Sin05], Sinz et al. present $LT^{n,k}_{SEQ}$, also known as the sequential counter encoding, to deal with cardinality constraints in the form $\leq k(A_1, ..., A_n)$. A sequential counter circuit of size n is composed by n subcircuits, each of which computes a partial sum $S_i = \sum_{j=1}^{i} A_j$, represented in unary form with the bits $S_{i,j}$, i.e., $S_{i,j} = \top$ if $\sum_{r=1}^{i} A_r \geq j$, so that $out_j \stackrel{\text{def}}{=} S_{n,j}$, $j \in [1..n]$. The (CNF version of the) following formula is the encoding of $LT^{n,k}_{SEQ}$ presented in [Sin05], for $k \stackrel{\text{def}}{=} n$:

$$(A_1 \rightarrow S_{1,1}) \wedge \textstyle\bigwedge_{i=2}^{n}\{((A_i \vee S_{i-1,1}) \rightarrow S_{i,1})\} \wedge \tag{4.7}$$

$$\textstyle\bigwedge_{i=2}^{n}\{(\neg A_i \vee \neg S_{i-1,n})\} \wedge \bigwedge_{j=2}^{n}\{(\neg S_{1,j})\} \wedge \tag{4.8}$$

$$\textstyle\bigwedge_{i,j=2}^{n}\{(((A_i \wedge S_{i-1,j-1}) \vee S_{i-1,j}) \rightarrow S_{i,j})\} \tag{4.9}$$

To make it bidirectional, we reintroduced the left implications "$\leftarrow$" of the encoding of each gate that were dropped in [Sin05]:

$$(A_1 \leftarrow S_{1,1}) \wedge \textstyle\bigwedge_{i=2}^{n}\{((A_i \vee S_{i-1,1}) \leftarrow S_{i,1})\} \wedge \tag{4.10}$$

$$\textstyle\bigwedge_{i,j=2}^{n}(((A_i \wedge S_{i-1,j-1}) \vee S_{i-1,j}) \leftarrow S_{i,j}). \tag{4.11}$$

The main advantage of the bidirectional sequential counter encoding is that it is fully incremental, meaning that a sorting circuit C with n inputs $A_1, ..., A_n$ can be easily (1) extended with an extra unary adder subcircuit to deal with $n+1$ inputs $A_1, ..., A_{n+1}$ (2) shrunk by dropping the last subcircuit to deal with the $n-1$ inputs $A_1, ..., A_{n-1}$ only. This property is particularly useful when dealing with $\mathcal{PB}$/MAXSMT objectives defined via the `assert-soft` command (see Section §5.3.1), since "soft" clauses can be incrementally pushed and popped from the formula stack. On the other hand, $LT_{SEQ}^{n,k}$ requires $O(k \cdot n)$ clauses and variables, and since in our application $k \stackrel{\text{def}}{=} n$ then it can be resource-demanding when dealing with a large number of inputs.

Bidirectional Cardinality Network Encoding [ANORC11, ANOR13]. The cardinality network encoding, presented in [ANORC11, ANOR13], is based on the underlying sorting scheme of the well-known *merge-sort* algorithm. The main advantage of the cardinality network encoding is that its space complexity is bounded by $O(n \log^2 k)$ in the number of clauses and variables where, in our case, $k = n$.

In this dissertation, we omit an illustration of this encoding due to the fact that its technical details are not quite relevant to the presentation as a whole, and also because we would not be able to add anything interesting on top of the very detailed examination made by the original authors in [ANORC11, ANOR13]. We refer the interested reader to these publications.

4.2.2 MAXRES Approach [NB14, BP14]

Full Disclosure. MAXRES is a core-based MAXSAT engine that was first presented in [NB14] and later ported into Z3 for dealing with MAXSMT objectives [BP14]. Neither of these publications have been co-authored by the Ph.D. candidate. The engine treats both hard and soft clauses as hard clauses, and progressively relaxes the resulting problem by replacing all the soft clauses in the unsatisfiable cores found along the search with a fewer number of new soft clauses. The algorithm ends either when the unsatisfiable core contains no soft clause, meaning that the formula is unsatisfiable as a whole, or when a satisfiable solution is found, that is guaranteed to also be optimal by construction.

We describe a MAXRES implementation, built on [NB14, BP14], that was subsequently introduced in OPTIMATHSAT for dealing with both MAXSMT and generic Pseudo-Boolean objectives [ST17]. Our goal is to make our presentation as unique as possible, even when it is based on other people's work. On this regard, we note that the procedures in [NB14] focus on propositional satisfiability, whereas [BP14] does not include any pseudocode and it provides,

function MAXRES(φ_h, φ_s)

1: $\varphi_s,\ opt \coloneqq$ PREPROCESS(φ_s)
2: **while** $true$ **do**
3: $\quad\langle res, \mathcal{M}\rangle \coloneqq$ SMT.CHECK($\varphi_h \wedge \bigwedge_{\langle C_i, w_i\rangle \in \varphi_s} C_i$)
4: $\quad$**if** $(res ==$ UNKNOWN$)$ **then**
5: $\quad\quad$**return** $\langle$UNKNOWN$, -, \emptyset\rangle$
6: $\quad$**else if** $(res ==$ SAT$)$ **then**
7: $\quad\quad$**return** $\langle$SAT$, opt, \mathcal{M}\rangle$
8: $\quad$**else**
9: $\quad\quad\tau \coloneqq$ SMT.GET_UNSAT_CORE()
10: $\quad\quad\tau_s, w_{min} \coloneqq$ GET_SOFT_CORE(φ_s, τ)
11: $\quad\quad$**if** $(\tau_s == \emptyset)$ **then**
12: $\quad\quad\quad$**return** $\langle$UNSAT$, -, \emptyset\rangle$
13: $\quad\quad$**else**
14: $\quad\quad\quad\varphi_h \coloneqq\ \varphi_h\ \cup\ \bigvee_{\langle C_i, w_i\rangle \in \tau_s} \neg C_i$
15: $\quad\quad\quad opt \coloneqq opt + ($MINIMIZE $?\ w_{min}\ :\ -w_{min})$
16: $\quad\quad\quad\varphi_s \coloneqq\ \varphi_s\ \setminus \tau_s$
17: $\quad\quad\quad\varphi_s \coloneqq\ \varphi_s\ \cup \bigcup_{\langle C_i, w_i\rangle \in \tau_s} .(w_i - w_{min} > 0\ ?\ \langle C_i, w_i - w_{min}\rangle\ :\ \emptyset)$
18: $\quad\quad\quad$**if** $(\mid \tau_s \mid >\ 1)$ **then**
19: $\quad\quad\quad\quad\varphi_h \coloneqq\ \varphi_h\ \cup \bigcup_{\langle C_i, w_i\rangle \in \tau_s} .B_i \rightarrow (B_{i-1} \wedge C_i)$
$\quad\quad\quad\quad\quad$// $B_0 \coloneqq \top$, $\forall_{i>0}.B_i$ is fresh Boolean var
20: $\quad\quad\quad\quad\varphi_s \coloneqq\ \varphi_s\ \cup \bigcup_{\langle C_i, w_i\rangle \in \{\tau_s \setminus \langle C_1, w_1\rangle\}} .\langle B_{i-1} \vee C_i, w_{min}\rangle$

Figure 4.4: The MAXRES engine implemented in OPTIMATHSAT.

instead, a high-level description of the algorithm. We provide the missing pseudocode and we also (1) consider the case of soft clauses with arbitrary weights (i.e. negative, zero or positive valued) and (2) allow for both the minimization and the maximization of the objective function. Last, we note that the argument for the termination of MAXRES given in [BP14] assumes that each soft clause has an unitary weight. Although this assumption does not result in any loss of generality, we chose to show a variant of such proof that does not rely on it.

Input. The algorithm, shown in Figure 4.4, takes as input a set of hard clauses φ_h, that must always satisfied, and a set of n soft clauses $\varphi_s = \{\langle C_1, w_1\rangle, ..., \langle C_n, w_n\rangle\}$.

Preprocessing. The first step is to transform the input problem into a canonical representation (line 1). Any soft clause $\langle C_i, w_i \rangle$ with a weight w_i equal to 0 is removed from the set φ_s. Any soft clause $\langle C_i, w_i \rangle$ such that $w_i < 0$ is replaced by a new soft clause $\langle \neg C_i, -w_i \rangle$. To ensure that the optimal value of the objective function is preserved by this transformation, the negative weight w_i is added to an internal accumulator. Additionally, if the optimization goal is to maximize the objective function, then every soft clause $\langle C_i, w_i \rangle$ is replaced by $\langle \neg C_i, w_i \rangle$ and the corresponding weight w_i is also added to the internal accumulator. At the end of this phase, the set φ_s contains only positive-weighted soft clauses. The updated set of soft clauses and the value of the internal accumulator are then returned to the main function. The latter value is used to initialize the objective function, and it corresponds to the value of obj when the conjunction of φ_h with all the soft clauses in φ_s, treated as hard clauses, is satisfiable.

MAXRES main loop. The loop starts by checking the satisfiability of the conjunction of φ_h with all soft clauses in φ_s, treated as hard clauses (line 3). If the underlying SMT solver is unable to decide satisfiability, the algorithm stops with UNKNOWN (lines 4-5). If the conjunction of all formulas is satisfiable, then the search has terminated and the model $\mathcal{M}$ can be returned, together with the optimal value of the objective function *opt* (lines 6-7).

Otherwise, the SMT solver answers with UNSAT. The corresponding unsat core τ, which is such that $\tau \subseteq \{\varphi_h \cup \bigcup_{\langle C_i, w_i \rangle \in \varphi_s} C_i\}$ and $\tau \models_{\mathcal{T}} \bot$, is retrieved from the SMT solver (line 9). The function GET_SOFT_CORE() is invoked to retrieve the subset τ_s of all soft clauses contained in τ, and to get the minimum weight w_{min} among the weights of all soft clauses contained in τ_s (line 10). Here, w_{min} corresponds to the progress made by this search step towards the optimal value of the objective function and, by construction, it is larger than 0 when τ_s is different from the empty set. If τ_s is empty, then the conflict set τ only contains hard clauses and the search can be terminated as the input problem is unsatisfiable (lines 11-12). Otherwise, the conflict is caused by one or more soft clauses being asserted as hard. Therefore, the execution jumps at line 14 to relaxate the problem with the application of the *Maximum Resolution* rule as in [BP14].

The first step is to ensure that the unsat core τ cannot be generated again by learning the blocking clause $\bigvee_{\langle C_i, w_i \rangle \in \tau_s} \neg C_i$ (line 14). Any hard clause in the conflict set τ need not to be included in the blocking clause, as it is forcibly assumed when the SMT solver is invoked. As a result, the learned clause forces at least one soft clause $\langle C_i, w_i \rangle$ in τ_s to be assigned to $false$ in future iterations of the search, ensuring a small search progress towards the optimal solution. This progress is taken into account by compensating the value of the objective function according to the optimization direction (line 15). Moreover, since at each loop iteration all soft clauses are treated as hard ones, the entire soft core τ_s is removed from φ_s, as it would

otherwise cause a conflict when conjoined with the newly learned blocking clause (line 16). When some soft clause $\langle C_i, w_i \rangle$ in τ_s has weight $w_i > w_{min}$, the removal of $\langle C_i, w_i \rangle$ has to be compensated with the introduction of a new soft clause $\langle C_i, w_i' \rangle$ where $w_i' := w_i - w_{min}$ (line: 17). This preserves the contribution of the original soft clause $\langle C_i, w_i \rangle$ to the value of the objective function, even after it is removed from φ_s, when C_i is set to $false$.

When τ_s contains more than one soft clause, then additional *compensation clauses* are necessary to preserve the value of the objective function. The reason for this is that the transformations in lines 14-17 do not take into account that in future iterations of the optimization search multiple C_i from τ_s might be contemporarily assigned to $false$. In such a case, the value of the objective function should be compensated for a multiple $m \cdot w_{min}$ of w_{min}, where the factor m is given by the number of clauses C_i set to $false$. The code in lines 19-20 does exactly that. First, it introduces k fresh support variables B_i of Boolean type, one for each soft clause in τ_s, and constrains each B_i to imply the first $i-1$ clauses C_i being assigned $true$ (line 19). Then, it extends φ_s with $k-1$ new soft clauses of the form $C_i' := \langle B_{i-1} \vee C_i, w_{min} \rangle$. Intuitively, B_{i-1} is used to ensure that a soft clause C_i' which contains a C_i assigned to $false$ is ignored, unless another C_j with $j < i$ assigned to $false$ is known to exist. The conflict clause learned at line 14 guarantees the existence of some C_j assigned to $false$, and since this was already accounted for with the adjustment of the objective function value at line 15, it must not be handled again.

Termination. An argument for termination can be made by proving that the innermost part of the loop, comprised by the lines ranging from 14 up to 20, can only be executed for a finite number of times. Given this fact, it is then trivial to see that the algorithm is subsequently forced to terminate at lines 5, 7 or 12.

Hence, we limit our discussion to the case in which MAXRES found a non-empty conflict set of soft clauses τ_s. Let ϕ be a set of soft clauses in the form $\{\langle C_1, w_1 \rangle, ..., \langle C_n, w_n \rangle\}$), then we define $\mathrm{W}(\phi)$ to be a function yielding the total weight of all soft clauses contained in ϕ, i.e. $\sum_{i=1}^{i=n} w_i$. The key observation of this proof is that the set of soft clauses τ_s is first removed from φ_s (line 16) and then replaced by a new set of soft clauses τ_s' (lines 17/20) such that $\mathrm{W}(\tau_s') < \mathrm{W}(\tau_s)$. This can be seen by expanding the expression of $\mathrm{W}(\tau_s)$

$$\begin{aligned} \mathrm{W}(\tau_s) &\stackrel{\text{def}}{=} \mathrm{W}(\{\langle C_1, w_1 \rangle, ..., \langle C_{|\tau_s|}, w_{|\tau_s|} \rangle\}) \\ &= \sum_{i=1}^{i=|\tau_s|} w_i = \sum_{i=1}^{i=|\tau_s|} (w_{min} + w_i - w_{min}) \\ &= \sum_{i=1}^{i=|\tau_s|} (w_i - w_{min}) + \mid \tau_s \mid \cdot w_{min} \end{aligned} \tag{4.12}$$

and the one of $\mathrm{w}(\tau'_s)$

$$\begin{aligned}
\mathrm{w}(\tau'_s) &\stackrel{\text{def}}{=} \mathrm{w}\big(\bigcup\nolimits_{\langle C_i,w_i\rangle\in\tau_s}.\langle C_i, w_i - w_{min}\rangle\big) \\
&\quad + \mathrm{w}\big(\bigcup\nolimits_{\langle C_i,w_i\rangle\in\{\tau_s\setminus\langle C_1,w_1\rangle\}}.\langle B_{i-1}\vee C_i, w_{min}\rangle\big) \\
&= \sum_{i=1}^{i=|\tau_s|}(w_i - w_{min}) + \sum_{i=2}^{i=|\tau_s|} w_{min} \\
&= \sum_{i=1}^{i=|\tau_s|}(w_i - w_{min}) + (\mid \tau_s \mid -1)\cdot w_{min}
\end{aligned} \tag{4.13}$$

so that we get

$$\begin{aligned}
\mathrm{w}(\tau_s) - \mathrm{w}(\tau'_s) &\stackrel{\text{def}}{=} \big(\sum_{i=1}^{i=|\tau_s|}(w_i - w_{min}) + \mid \tau_s \mid \cdot w_{min}\big) \\
&\quad - \big(\sum_{i=1}^{i=|\tau_s|}(w_i - w_{min}) + (\mid \tau_s \mid -1)\cdot w_{min}\big) \\
&= \sum_{i=1}^{i=|\tau_s|}(w_i - w_{min}) - \sum_{i=1}^{i=|\tau_s|}(w_i - w_{min}) \\
&\quad + \mid \tau_s \mid \cdot w_{min} - (\mid \tau_s \mid -1)\cdot w_{min} \\
&= 0 + (\mid \tau_s \mid - \mid \tau_s \mid + 1)\cdot w_{min} \\
&= w_{min}
\end{aligned} \tag{4.14}$$

By construction, all soft clauses in φ_s have a strictly positive weight w_i. The value w_{min} is also guaranteed to be positive, since it is equal to the minimum weight w_i of all soft clauses in the conflict set $\tau_s \subseteq \varphi_s$. Hence, by Equation (4.14), we have that $\mathrm{w}(\tau'_s) < \mathrm{w}(\tau_s)$ and that the cumulative weight of the set of soft clauses $\varphi'_s \coloneqq \varphi_s \setminus \tau_s \cup \tau'_s$ resulting from the execution of lines 14-20 is strictly smaller than that of φ_s. Therefore, we can conclude that the algorithm can perform only a finite number of relaxations, stretching at most up until when $\mathrm{w}(\varphi'_s) \coloneqq 0$, and then it has to terminate.

Experimental Evaluation. In Section §6.2, we demonstrate the benefits of the sorting network circuit enhancement and the remarkable performance of the MAXRES engine with a number of experimental evaluations on OMT($\mathcal{PB} \cup \mathcal{T}$) and Partial Weighted MAXSMT formulas.

4.3 OMT ($\mathcal{BV} \cup \mathcal{T}$)

Full Disclosure. The *Bit-Vector Optimization with Weak Assumptions* (OBV-WA) and the *Bit-Vector Optimization with Binary Search* (OBV-BS) algorithms, described in Section §4.3.2 and Section §4.3.3 respectively, have been first presented by Nadel et al. in [NR16] and have not been co-authored by the Ph.D. candidate.

Unsigned Bit-Vector optimization was first introduced by Bjorner et al. in [BP14, BPF15], and was later drastically improved by Nadel et al. in [NR16]. In this thesis, we present a generalization of the methods and techniques described in [BP14, BPF15, NR16] to the case of both signed and unsigned $\mathcal{BV}$ optimization. We have presented this approach in [TS19].

Without any loss of generality, we assume that every non-trivial $\mathcal{BV}$ objective function $f(...)$ is replaced by a $\mathcal{BV}$ variable obj by conjoining "obj $= f(...)$" to the input formula. We use the symbol n to denote the bit width of obj, and obj$[i]$ to denote the i-th bit of obj, where obj$[0]$ and obj$[n-1]$ are the Most Significant Bit (MSB) and the Least Significant Bit (LSB) of obj respectively.[3] We use the symbol μ_k to denote a generic (possibly partial) assignment that assigns at least the k most-significant bits of obj. We use the symbol τ_k to denote an assignment to the k most-significant bits of obj. Given $i < k$, we denote by $\mu_k[i]$ [resp. $\tau_k[i]$] the value in $\{0,1\}$ assigned to obj$[i]$ by μ_k [resp. τ_k]. Moreover, we use the expression $[\![\mu_k]\!]_i$ where $i \leq k$ to denote the restriction of μ_k to the i most-significant bits of obj, obj$[0], ...,$ obj$[i-1]$. Given a model $\mathcal{M}$ of φ and a variable v, we denote by $\mathcal{M}(v)$ the evaluation of v in $\mathcal{M}$.

We define the *Bit-Vector Optimization problem* as follows.

Definition 4.3.1. *(OMT($\mathcal{BV} \cup \mathcal{T}$), OMT($\mathcal{BV}$), $\mathsf{min}_{\mathsf{obj}}$). Let φ be a ground SMT($\mathcal{BV} \cup \mathcal{T}$) formula and* obj *be a —signed or unsigned— $\mathcal{BV}$ variable occurring in φ. We call an Optimization Modulo $\mathcal{BV} \cup \mathcal{T}$ problem, the problem of finding a model $\mathcal{M}$ for φ (if any) whose value of* obj*, denoted with $\mathsf{min}_{\mathsf{obj}}(\varphi)$, is minimum with respect to the total order relation $\leq_n$ for signed $\mathcal{BV}$s if* obj *is signed, and the one for unsigned $\mathcal{BV}$s otherwise. We call an Optimization Modulo $\mathcal{BV}$ problem, written OMT($\mathcal{BV}$), an OMT($\mathcal{BV} \cup \mathcal{T}$) problem where $\mathcal{T}$ is the empty theory. (The dual definition where we look for the maximum follows straightforwardly)*

We introduce the (novel) notion of a $\mathcal{BV}$ *attractor* to generalize the unsigned $\mathcal{BV}$ optimization methods described in [BP14, BPF15, NR16] to the case of signed and unsigned $\mathcal{BV}$ optimization.

Definition 4.3.2. *(Attractor, Attractor equalities). When minimizing [resp. maximizing], we call attractor for* obj *the smallest [resp. greatest] $\mathcal{BV}$-value $attr$ of the sort of* obj*. We call vector of attractor equalities the vector A such that $A[k] \stackrel{def}{=} (\mathsf{obj}[k] = attr[k])$, $k \in [0..n-1]$.*

Example 4.3.1. *If* $\mathsf{obj}^{[8]}$ *is an* unsigned $\mathcal{BV}$ *objective of width* 8*, then its corresponding attractor $attr$ is $\mathbf{0}^{[8]}$, i.e. $[00000000]$, when* $\mathsf{obj}^{[8]}$ *is minimized and it is $\mathbf{255}^{[8]}$, i.e. $[11111111]$, when* $\mathsf{obj}^{[8]}$ *is maximized. When* $\mathsf{obj}^{[8]}$ *is instead a* signed $\mathcal{BV}$ *objective, following the two's complement encoding, the corresponding $attr$ is $-\mathbf{128}^{[8]}$, i.e. $[10000000]$, for minimization and $\mathbf{127}^{[8]}$, i.e. $[01111111]$, for maximization.* ◇

In essence, the *attractor* can be seen as the target value of the optimization search and therefore it can be used to determine the desired improvement direction and to guide the decisions taken by the optimization search. By construction, if a model $\mathcal{M}$ satisfies all equalities $A[i]$, then $\mathcal{M}(\mathsf{obj}) = attr$. More generally, if $\mathcal{M}$ is a model of φ, then the value of obj in $\mathcal{M}$, denoted with $\mathcal{M}(\mathsf{obj})$, is given by

$$\tau(\mathsf{obj}) = \sum_{i=0}^{i=n-1} (2^{n-1-i} \cdot \text{ITE}(\mathcal{M}(A[i]), attr[i], \overline{attr[i]})) \tag{4.15}$$

when obj is an *unsigned* $\mathcal{BV}$ objective, and by

$$\begin{aligned} \tau(\mathsf{obj}) = & \sum_{i=1}^{i=n-1} (2^{n-1-i} \cdot \text{ITE}(\mathcal{M}(A[i]), attr[i], \overline{attr[i]})) \\ & - (2^{n-1}) \cdot \text{ITE}(\mathcal{M}(A[0]), attr[0], \overline{attr[0]}) \end{aligned} \tag{4.16}$$

when obj is a *signed* $\mathcal{BV}$ objective, using the *two's complement* representation. The function ITE, appearing in both previous equations, returns $attr[i]$ if the *attractor equality* $A[i]$ is true in $\mathcal{M}$ and $\overline{attr[i]}$ otherwise.

With a small abuse of notation, and when this does not cause ambiguities, we sometimes use an attractor equality $A[i] \stackrel{\text{def}}{=} (\mathsf{obj}[i] = attr[i])$ to denote the single-bit assignment $\mathsf{obj}[i] := attr[i]$ and its negation $\neg A[i]$ to denote the assignment to the complement value $\mathsf{obj}[i] := \overline{attr[i]}$.

Definition 4.3.3. *(Lexicographic maximization) Consider an OMT instance $\langle \varphi, \mathsf{obj} \rangle$ and the vector of attractor equalities A. We say that an assignment τ_n to* obj l*exicographically maximizes A with respect to φ if and only if, for every $k \in [0..n-1]$,*

- $\tau_n[k] = \overline{attr[k]}$ *if $\varphi \wedge [\![\tau_n]\!]_k \wedge A[k]$ is unsatisfiable,*
- $\tau_n[k] = attr[k]$ *otherwise.*

where $A[k]$ is the attractor equality $(\mathsf{obj}[k] = attr[k])$. *(The dual definition of* "lexicographically minimizes" *switches $attr[k]$ with $\overline{attr[k]}$.) Given a model $\mathcal{M}$ for φ, we say that $\mathcal{M}$ lexicographically maximizes A with respect to φ if and only if its restriction to* obj *lexicographically maximizes A with respect to φ.*

Starting from the MSB to the LSB, τ_n [resp. $\mathcal{M}$] in Definition 4.3.3 assigns to each $\mathsf{obj}[k]$ the value $attr[k]$ unless it is inconsistent with respect to φ and the assignments to the previous $\mathsf{obj}[i]$s, $i \in [0..k-1]$. Notice that this corresponds to minimize [resp. maximize] the value $\sum_{k=0}^{n-1} 2^{n-1-k} \cdot (\mathsf{obj}[k] \mathbf{xor}_1 attr[k])$ [resp. $\sum_{k=0}^{n-1} 2^{n-1-k} \cdot (\mathsf{obj}[k]\ \mathbf{nxor}_1\ attr[k])$], —where $\mathbf{xor}_n$ is the bitwise-xor operator and $\mathbf{nxor}_n$ is its complement— because $2^{n-1-i} > \sum_{k=i+1}^{n-1} 2^{n-1-k}$.

Example 4.3.2. *Let* $\mathsf{obj}^{[3]}$ *be a signed $\mathcal{BV}$ goal of 3 bits to be minimized and $attr \stackrel{def}{=} [100]$ be its attractor, so that the corresponding vector of attractor equalities A is equal to* $[\mathsf{obj}[0] = 1, \mathsf{obj}[1] = 0, \mathsf{obj}[2] = 0]$.

An assignment $\tau_3 \stackrel{def}{=} \{A[0], \neg A[1], \neg A[2]\}$ (for which $\mathsf{obj}^{[3]} = -\mathbf{1}^{[3]}$) is lexicographically better than $\tau_3' \stackrel{def}{=} \{\neg A[0], A[1], A[2]\}$ (for which $\mathsf{obj}^{[3]} = \mathbf{0}^{[3]}$), because the former satisfies the attractor equality *corresponding to the MSB while the latter does not. Moreover, the assignment τ_3 is lexicographically worse than the assignment $\tau_3'' \stackrel{def}{=} \{A[0], \neg A[1], A[2]\}$ (for which $\mathsf{obj}^{[3]} = -\mathbf{2}^{[3]}$), because —all the rest being equal— the latter assignment makes the* attractor equality $(\mathsf{obj}[2] = 0)$ *true.* ◇

The following fact derives from the above definitions and the properties of two's complement representation adopted by the SMT-LIBv2 standard[10] for signed $\mathcal{BV}$.

Theorem 4.3.4. An optimal solution of an OMT($\mathcal{BV} \cup \mathcal{T}$) problem $\langle \varphi, \mathsf{obj} \rangle$ is any model $\mathcal{M}$ of φ that *lexicographically maximizes* the vector of *attractor equalities* A.

We demonstrate that Theorem 4.3.4 holds, for both signed and unsigned $\mathcal{BV}$ objectives, using the following argument.

Proof. (We prove the case of minimization, since that of maximization is dual)

Let $\langle \varphi, \mathsf{obj} \rangle$ be an OMT($\mathcal{BV} \cup \mathcal{T}$) problem where obj is a $\mathcal{BV}$ objective to be minimized and $attr$ is its attractor, that is, the smallest value that can be represented with a $\mathcal{BV}$ value of the same sort of obj. Let A be the corresponding vector of attractor equalities such that, for each i, $A[i] = (\mathsf{obj}[i] = attr[i])$. Finally, let $\mathcal{M}$ be a a model of φ that *lexicographically maximizes* A, where τ_n is the restriction of $\mathcal{M}$ to the n bits of obj and $\mathcal{M}(\mathsf{obj})$ is the model value of obj, computed with Equation (4.15) if obj is an *unsigned*

[10] If the standard adopted were the sign-and-magnitude binary encoding, then Theorem 4.3.4 would not hold. Nevertheless, in such a case we could adopt a simplified version of the technique for $\mathcal{FP}$ optimization described in Section §4.4.

$\mathcal{BV}$ objective and with Equation (4.16) otherwise. By definition, model $\mathcal{M}$ is an optimal solution for $\langle \varphi, \mathsf{obj} \rangle$ if there exists no other model $\mathcal{M}'$ such that $\mathcal{M}'$ satisfies φ and $\mathcal{M}'(\mathsf{obj}) < \mathcal{M}(\mathsf{obj})$. Hence, we show by contradiction that no such $\mathcal{M}'$ can exist.

Assume (for the sake of contradiction), that there exists a model $\mathcal{M}'$ of φ such that $\mathcal{M}'(\mathsf{obj}) < \mathcal{M}(\mathsf{obj})$, and let τ'_n be the restriction of $\mathcal{M}'$ to the n bits of obj. Since $\mathcal{M}'(\mathsf{obj}) \neq \mathcal{M}(\mathsf{obj})$, by Equations (4.15)-(4.16) there must be at least one index i for which $\tau_n[i] \neq \tau'_n[i]$. Let m be the smallest index i, from 0 to $n-1$, such that $\tau_n[m] \neq \tau'_n[m]$. We set $\tau_m \stackrel{\text{def}}{=} [\![\tau_n]\!]_m$, $\tau_{m+1} \stackrel{\text{def}}{=} [\![\tau_n]\!]_{m+1}$ and $\tau'_{m+1} \stackrel{\text{def}}{=} [\![\tau'_n]\!]_{m+1}$. Then, $\tau_m \subset \tau_{m+1}$, $\tau_m \subset \tau'_{m+1}$, $\tau_{m+1} \neq \tau'_{m+1}$. In particular, $\tau_{m+1}[m] = \overline{\tau'_{m+1}[m]}$ and therefore $\tau_{m+1}[m] = attr[m]$ if $\tau'_{m+}[m] = \overline{attr[m]}$, and vice versa.

We split our proof in two parts: in the first we consider of an *unsigned* $\mathcal{BV}$ goal, while in the second part we consider the case of a *signed* $\mathcal{BV}$ objective.

CASE I: obj **is an unsigned** $\mathcal{BV}$ **goal.** Then, we distinguish two subcases.

In the first case, $\tau_{m+1}[m] = \overline{attr[m]}$ and $\tau'_{m+1}[m] = attr[m]$. From $\tau_{m+1}[m] = \overline{attr[m]}$ and the fact that τ_n lexicographically maximizes A, we derive that $\varphi \wedge \tau_m \wedge A[m]$ is unsatisfiable, where $A[m] \stackrel{\text{def}}{=} (\mathsf{obj}[m] = attr[m])$. Since $\tau_m \subset \tau'_{m+1} \subseteq \tau'_n$ and $\tau'_{m+1}[m] = attr[m]$, we conclude that $\varphi \wedge \tau'_n$ is unsatisfiable, so that $\mathcal{M}'$ cannot be a model of φ, contradicting the initial assumption.

In the second case, $\tau_{m+1}[m] = attr[m]$ and $\tau'_{m+1}[m] = \overline{attr[m]}$. For $\mathcal{M}'(\mathsf{obj}) < \mathcal{M}(\mathsf{obj})$ to hold, the difference $\Delta \stackrel{\text{def}}{=} \mathcal{M}'(\mathsf{obj}) - \mathcal{M}(\mathsf{obj})$ computed in Equation (4.17) must be smaller than zero.

$$
\begin{aligned}
\Delta \;=\; & \sum_{i=0}^{i=n-1} (2^{n-1-i} \cdot \textsc{ite}(\mathcal{M}'(A[i]), attr[i], \overline{attr[i]})) - \\
& \sum_{i=0}^{i=n-1} (2^{n-1-i} \cdot \textsc{ite}(\mathcal{M}(A[i]), attr[i], \overline{attr[i]})) \\
=\; & 2^{n-1-m} \cdot (\overline{attr[m]} - attr[m]) + \\
& \sum_{i=m+1}^{i=n-1} (2^{n-1-i} \cdot \textsc{ite}(\mathcal{M}'(A[i]), attr[i], \overline{attr[i]})) - \\
& \sum_{i=m+1}^{i=n-1} (2^{n-1-i} \cdot \textsc{ite}(\mathcal{M}(A[i]), attr[i], \overline{attr[i]})) \\
=\; & 2^{n-1-m} + \\
& \sum_{i=m+1}^{i=n-1} (2^{n-1-i} \cdot (\textsc{ite}(\mathcal{M}'(A[i]), attr[i], \overline{attr[i]}) - \textsc{ite}(\mathcal{M}(A[i]), attr[i], \overline{attr[i]}))
\end{aligned}
\tag{4.17}
$$

In Equation (4.17), we have used the following facts: (I) $\mathcal{M}(A[k]) = \mathcal{M}'(A[k])$ for every

$k \in [0, m-1]$, (II) $\mathcal{M}(A[m])$ is true, (III) $\mathcal{M}'(A[m])$ is false and (IV) $\overline{attr[m]} - attr[m]$ is equal 1 because obj is interpreted as an unsigned $\mathcal{BV}$ and it is being minimized, which means that $attr = \mathbf{0}^{[n]}$.

It can be seen that there cannot exist any pair of models $\langle \mathcal{M}, \mathcal{M}' \rangle$ such that $\mathcal{M}'[\mathsf{obj}] < \mathcal{M}[\mathsf{obj}]$, because for any positive $k \stackrel{\text{def}}{=} n - 1 - m$, the value 2^k (a.k.a. 2^{n-1-m}) is strictly larger than $\sum_{i=0}^{i=k-1} 2^i$ (a.k.a. $\sum_{i=m+1}^{i=n-1} 2^{n-1-i}$). This contradicts the initial assumption that $\mathcal{M}'(\mathsf{obj}) < \mathcal{M}(\mathsf{obj})$.

CASE II: obj is a signed $\mathcal{BV}$ goal. Then, we distinguish —once again— two subcases.

In the first case, m is larger than 0. Therefore, $\mathcal{M}(\mathsf{obj})$ and $\mathcal{M}'(\mathsf{obj})$ are either both negative or both positive. Looking at Equation (4.16), which adheres to the *two's complement* representation rules, we observe that the *sign bit* results in a constant-valued displacement of the model value $\mathcal{M}(\mathsf{obj})$, that is otherwise equal to the one computed by Equation (4.15). Since the value assigned to $\mathsf{obj}[0]$ by $\mathcal{M}$ and $\mathcal{M}'$ provides the same (fixed) contribution to the value of obj, we can ignore the *sign bit* and restrict our focus on the remaining $n-1$ bits. In this way, we obtain a new $\mathsf{obj}' \stackrel{\text{def}}{=} [\mathsf{obj}[1], ..., \mathsf{obj}[n-1]]$, a new $attr' \stackrel{\text{def}}{=} [attr[1], ..., attr[n-1]]$ and a new $A' \stackrel{\text{def}}{=} [(\mathsf{obj}[1] = attr[1]), ..., (\mathsf{obj}[n-1] = attr[n-1])]$. For all intents and purposes, following the *two's complement* representation, the new goal obj' can be considered as an *unsigned* $\mathcal{BV}$ goal as it was obtained by dropping the *sign bit* from obj. As a consequence, we can apply the same argument illustrated in **CASE I** over the new obj', $attr'$ and A'.

In the second case, m is equal to 0. By assumption, $\mathcal{M}'(\mathsf{obj}) < \mathcal{M}(\mathsf{obj})$ so we have that $\mathcal{M}'(\mathsf{obj}) < 0$ and $\mathcal{M}(\mathsf{obj}) \geq 0$. Given that obj is being minimized, the value of $attr[0]$ is equal to 1, and therefore we immediately get that $[\![\tau_n]\!]_1[0] = \overline{attr[0]}$ and $[\![\tau'_n]\!]_1[0] = attr[0]$. From $[\![\tau_n]\!]_1[0] = \overline{attr[0]}$ and the fact that, by assumption, $\mathcal{M}$ —hence τ_n— lexicographically maximizes A, we have that $\varphi \wedge [\![\tau_n]\!]_0 \wedge A[0]$ is unsatisfiable. Since $[\![\tau_n]\!]_0 = \emptyset$ and $A[0] \stackrel{\text{def}}{=} (\mathsf{obj}[0] = attr[0])$, then it follows that $\varphi \wedge (\mathsf{obj}[0] = attr[0])$ is unsatisfiable. Therefore, we conclude that $\mathcal{M}'$ cannot be a satisfiable model of φ, contradicting the initial assumption. □

Using Definitions 4.3.2 and 4.3.3 with Theorem 4.3.4, we can reduce any *Bit-Vector Optimization problem* $\langle \varphi, \mathsf{obj} \rangle$ to a lexicographic OMT($\mathcal{LRA} \cup \mathcal{T}$) problem or to partial weighted MAXSMT.

Reduction to lexicographic OMT($\mathcal{LRA} \cup \mathcal{T}$). A OMT($\mathcal{BV} \cup \mathcal{T}$) problem can be straightforwardly encoded into a lexicographic OMT($\mathcal{LRA} \cup \mathcal{T}$) instance $\langle \varphi, \mathsf{obj}_0, ..., \mathsf{obj}_{n-1} \rangle$ by setting each obj_i equal to $\text{ITE}(A[i], 1, 0)$, where $A[i]$ is the i-th attractor equality $(\mathsf{obj}[i] = attr[i])$. With this approach, the maximization of each objective function in lexicographic order results

in the lexicographical satisfaction of the set of *attractor equalities*. Therefore, by Proposition 4.3.4, the optimal solution of the original OMT($\mathcal{BV} \cup \mathcal{T}$) problem is returned.

Reduction to partial weighted MAXSMT [BP14, BPF15].[11] A OMT($\mathcal{BV} \cup \mathcal{T}$) problem can be encoded as a partial weighted MAXSMT problem $\langle \varphi_h, \varphi_s \rangle$, where the input formula φ is transformed into the set of "hard" $\mathcal{T}$-clauses φ_h, and A is assigned to the set of "soft" $\mathcal{T}$-clauses φ_s. The weight $\mathbf{a}_i$ of each each soft clause $(\mathsf{obj}[i] = attr[i])$, is set to be equal to 2^{n-1-i}. We recall here that the goal of a partial weighted MAXSMT problem is to find the subset of "soft" $\mathcal{T}$-clauses with maximum-weight, as described in §2.3.3. Since the weight $\mathbf{a}_i$ of each "soft" $\mathcal{T}$-clause C_i is strictly larger than the sum of the weights of all C_j such that $j > i$, the MAXSMT solver assigns higher preference to the satisfaction of the attractor equality $(\mathsf{obj}[i] = attr[i])$ than to any other attractor equality $(\mathsf{obj}[j] = attr[j])$ where $j > i$. Thus, the optimal model found by the partial weighted MAXSMT search is also optimal for the original OMT($\mathcal{BV} \cup \mathcal{T}$) problem by Proposition 4.3.4.

Rather than using a reduction to a known problem, it is possible to approach OMT($\mathcal{BV} \cup \mathcal{T}$) more directly using a dedicated engine for $\mathcal{BV}$ optimization. In the following, we consider three main approaches for OMT($\mathcal{BV} \cup \mathcal{T}$). Section §4.3.1 illustrates the case of a simple linear- and binary-search approach, based on the inline OMT schema presented in §2.3.1. Sections §4.3.2 and §4.3.3 present the signed extensions of the *Bit-Vector Optimization with Weak Assumptions* (OBV-WA) and the *Bit-Vector Optimization with Binary Search* (OBV-BS) algorithms respectively, both of which have been originally presented by Nadel et al. in [NR16]. For these last two approaches, we provide a concise description of the main ideas underlying these methods and then proceed by illustrating the existing differences among the procedures described in [NR16] and our re-implementation in OPTIMATHSAT.

4.3.1 Signed/unsigned OMT-based Search

The OMT-based schema presented here is a new extension of the inline linear-, binary- and adaptive-search schemas for OMT($\mathcal{LRA} \cup \mathcal{T}$) —first presented in [ST12, ST15a] and described in Section §2.3.1— to deal with both signed and unsigned $\mathcal{BV}$ objectives. In this regard, we note that in [NR16] Nadel et al. describe a simple linear- and binary-schema for unsigned $\mathcal{BV}$ optimization built on top of an SMT solver used as a black-box. However, in contrast with [NR16], in OPTIMATHSAT the optimization schema is *inline* with the underlying SMT solver.

[11] We generalize the unsigned $\mathcal{BV}$ optimization approach employed by [BP14, BPF15] to the case of both signed and unsigned $\mathcal{BV}$ optimization using the notion of $\mathcal{BV}$ objective *attractor*. Furthermore, we note that the original approach is described in [NR16], that cites *"private communication with the authors of [BP14, BPF15]"* as its own source, and not in [BP14, BPF15].

This approach, implemented on top of OPTIMATHSAT, has been shown to outperform the corresponding *offline* implementation on a set of OMT($\mathcal{LRA} \cup \mathcal{T}$) benchmarks included in an experimental evaluation of [ST12, ST15a].

On the whole, very few changes are necessary to adapt the OMT schemas for OMT($\mathcal{LRA} \cup \mathcal{T}$) (see Section §2.3.1) to the case of signed and unsigned $\mathcal{BV}$ optimization.

First, we notice that the domain of any $\mathcal{BV}$ variable is finite in the Theory of Bit-Vectors, and thus —in principle— a simple enumeration of all possible values would suffice to exhaust the solution space and find the desired optimal solution, similarly to the case of a $\mathcal{LIA}$ objective with finite domain.

In the case of an *eager* $\mathcal{BV}$-encoding in particular, we observe that extending the $\mathcal{BV}$-solver with an embedded optimization procedure would be wasting CPU cycles. This is because $\mathcal{BV}$ terms are *bit-blasted* during the preprocessing of the formula, and therefore the value of any $\mathcal{BV}$ objective is distinctively determined by the truth assignment that is being generated by the CDCL engine. As a result, an optimization procedure confined within the $\mathcal{BV}$ engine would not be able to locally improve the value of the objective function.

When the *lazy* $\mathcal{BV}$-encoding is used, an optional dedicated optimization procedure can be embedded within the $\mathcal{BV}$-solver, although this is not strictly necessary. In the following, to keep things as simple as possible, we assume that no such dedicated optimization procedure is embedded within the Bit-Vector $\mathcal{T}$-solver.

Second, we notice that the size of the domain of an unconstrained $\mathcal{BV}$ variable of bit width n —albeit finite— is still exponential in n. This means that a simple linear search over a $\mathcal{BV}$ objective may end up enumerating an exponential number of satisfiable truth assignments before finding the optimal solution value. Therefore, it is pretty much always convenient to run the optimization in binary-search mode. In its simplest implementation, the binary-search strategy places the pivot value so as to halve the solution space, determined by the range of bits, at each iteration. Thus, the search is guaranteed to converge in at most n binary-search iterations, where n is the number of bits of the $\mathcal{BV}$ objective. An alternative approach is to use the following heuristic function, evaluated on the *Rational* domain and converted back to a $\mathcal{BV}$ value, to compute the pivot placement:

$$\mathsf{pivot} \stackrel{\text{def}}{=} \mathtt{floor}(\rho \cdot \mathsf{ub} + (1 - \rho) \cdot \mathsf{lb}) + \Delta$$

where lb and ub are the current lower and upper bound of obj respectively, ρ belongs to the interval $[0, 1[$ (e.g. $\frac{1}{2}$) and Δ is equal to 1 if the expression inside $floor()$ is fractional and the objective is to be minimized, 0 otherwise.

This heuristic has two advantages. First, it allows one to control the aggressiveness of the range-partitioning strategy by adjusting the value of ρ. Second, it takes into account any

optional initial *lower/upper bounds* provided by the user of the application. As shown in the following example, this can result in fewer search steps to converge to the optimal solution. We note, however, that fewer search steps do not necessarily mean better overall performance, as the cost of each binary-step is non-uniform and it usually cannot be predicted in advance.

Example 4.3.3. *Let* $\mathsf{obj}^{[8]}$ *be a* $\mathcal{BV}$ *objective of width* 8 *to be minimized, and let* $[\mathbf{50}^{[8]}, \mathbf{64}^{[8]}[$ *be the initial search interval specified by the end-user and* $\mathbf{61}^{[8]}$ *its optimal value. Assume, for simplicity, that each satisfiable step results in the smallest possible improvement of the objective function, and that the input formula admits a satisfiable model whenever the optimization goal is included in the range of values* $[\mathbf{61}^{[8]}, \mathbf{63}^{[8]}]$.

Then, the sequence of pivot *atoms generated by an OMT solver minimizing* $\mathsf{obj}^{[8]}$ *with a binary-search computed over the* bits *of the objective function is as follows:*

$\mathsf{obj}^{[8]} <_{[8]} \mathbf{128}^{[8]}$ *(sat,* $\mathcal{M}[\mathsf{obj}^{[8]}] = \mathbf{63}^{[8]}$*; s.i.:* $[\mathbf{50}^{[8]}, \mathbf{63}^{[8]}[$*)* $\Longrightarrow$ *continue*
$\mathsf{obj}^{[8]} <_{[8]} \mathbf{64}^{[8]}$ *(sat,* $\mathcal{M}[\mathsf{obj}^{[8]}] = \mathbf{62}^{[8]}$*; s.i.:* $[\mathbf{50}^{[8]}, \mathbf{62}^{[8]}[$*)* $\Longrightarrow$ *continue*
$\mathsf{obj}^{[8]} <_{[8]} \mathbf{32}^{[8]}$ *(unsat, s.i.:* $[\mathbf{50}^{[8]}, \mathbf{62}^{[8]}[$*)* $\Longrightarrow$ *backtrack*
$\mathsf{obj}^{[8]} <_{[8]} \mathbf{48}^{[8]}$ *(unsat, s.i.:* $[\mathbf{50}^{[8]}, \mathbf{62}^{[8]}[$*)* $\Longrightarrow$ *backtrack*
$\mathsf{obj}^{[8]} <_{[8]} \mathbf{56}^{[8]}$ *(unsat, s.i.:* $[\mathbf{56}^{[8]}, \mathbf{62}^{[8]}[$*)* $\Longrightarrow$ *backtrack*
$\mathsf{obj}^{[8]} <_{[8]} \mathbf{60}^{[8]}$ *(unsat, s.i.:* $[\mathbf{60}^{[8]}, \mathbf{62}^{[8]}[$*)* $\Longrightarrow$ *backtrack*
$\mathsf{obj}^{[8]} <_{[8]} \mathbf{62}^{[8]}$ *(sat,* $\mathcal{M}[\mathsf{obj}^{[8]}] = \mathbf{61}^{[8]}$*; s.i.:* $[\mathbf{60}^{[8]}, \mathbf{61}^{[8]}[$*)* $\Longrightarrow$ *continue*
$\mathsf{obj}^{[8]} <_{[8]} \mathbf{61}^{[8]}$ *(unsat, s.i.:* $[\mathbf{61}^{[8]}, \mathbf{61}^{[8]}[$*)* $\Longrightarrow$ *stop.*

In the above sequence, we show the optimization search interval (a.k.a "s.i."*) at the end of each binary-search step. The search, following the same schema described in Section §2.3.1, ends when the search interval becomes empty.*

If instead the OMT solver minimizes $\mathsf{obj}^{[8]}$ *with a binary-search computed over the actual* range *of the objective function, then the following sequence of* pivot *atoms is generated with* ρ *equal to* $\frac{1}{2}$*:*

$\mathsf{obj}^{[8]} <_{[8]} \mathbf{57}^{[8]}$ *(unsat, s.i.:* $[\mathbf{57}^{[8]}, \mathbf{63}^{[8]}[$*)* $\Longrightarrow$ *backtrack*
$\mathsf{obj}^{[8]} <_{[8]} \mathbf{61}^{[8]}$ *(unsat, s.i.:* $[\mathbf{61}^{[8]}, \mathbf{63}^{[8]}[$*)* $\Longrightarrow$ *backtrack*
$\mathsf{obj}^{[8]} <_{[8]} \mathbf{62}^{[8]}$ *(sat,* $\mathcal{M}[\mathsf{obj}^{[8]}] = \mathbf{61}^{[8]}$*; s.i.:* $[\mathbf{61}^{[8]}, \mathbf{61}^{[8]}[$*)* $\Longrightarrow$ *stop.*

Using this approach, the optimal solution is found with fewer search steps. $\diamond$

4.3.2 A signed extension of OBV-WA [NR16]

The *Bit-Vector Optimization with Weak Assumptions* algorithm, for the maximization of unsigned $\mathcal{BV}$ objectives, was first presented by Nadel et al. [NR16]. In their work, Nadel et al.

transformed the bits $[\mathsf{obj}[0], ..., \mathsf{obj}[n-1]]$ of the $\mathcal{BV}$ objective into high-priority decision variables for the underlying SAT solver, starting from the MSB down to the LSB. Moreover, they initialized the phase saving of each $\mathsf{obj}[i]$ to 1 (for minimization, to 0) to automatically drive the CDCL search towards the optimal value. Since the combination of these two techniques makes the algorithm approach the optimal value starting from the unsatisfiable region, the optimization search could be stopped as soon as a model was found.

We notice that this approach can be extended to deal with the optimization, in either direction, of both signed and unsigned $\mathcal{BV}$ objectives. To do so, we exploit the *attractor equalities* associated with the objective function obj.

First, for each *attractor equality* of the form $\mathsf{obj}[i] = attr[i]$ we introduce a fresh Boolean decision variable A_i such that $A_i \leftrightarrow \mathsf{obj}[i] = attr[i]$. Then, we provide the following (optional) enhancements over the regular OMT-based approach described in §4.3.1:

- **branching preference**: each A_i is added to the list of the preferred Boolean variables for branching, starting from the MSB down to the LSB. This has the benefit of causing the highest possible backjump when any conflict involving a unit clause in the form $(\mathsf{obj} < \mathsf{lb})$ [resp. $(\mathsf{obj} > \mathsf{ub})$] is encountered while minimizing [resp. maximizing] the objective function.
- **polarity initialization**: the phase-saving value of each A_i is initialized to true at the beginning of the search, so that the first time the SAT engine encounters A_i as a decision variable it tries to assign true first. As a result, the solver prefers looking for candidate values of obj that are closer to the target $attr$.

Activating both of these enhancements at the same time forces the OMT-based approach described in §4.3.1 to behave like the OBV-WA algorithm in [NR16] (disregarding the fact that the latter is designed to directly handle unsigned $\mathcal{BV}$ objectives only). This is because the Boolean search at the SAT level takes precedence over the outer optimization schema, and therefore the combination of the two enhancements makes the OMT solver approach the optimal solution starting from the (possibly empty) unsatisfiable region as in OBV-WA. As a consequence, the OMT solver does not need to certify that the value of the objective function is optimal and can return as soon as a model is found.

4.3.3 A signed extension of OBV-BS [NR16]

The *Bit-Vector Optimization with Binary Search* algorithm was also presented in [NR16] first. As for OBV-WA, the original algorithm dealt with the maximization of unsigned $\mathcal{BV}$ objectives

only, that we extend to also deal with signed $\mathcal{BV}$ objectives using the definition of $\mathcal{BV}$ objective *attractor* in Section §4.3.

The search starts by looking for an initial model $\mathcal{M}$ that satisfies the input formula φ. If no such $\mathcal{M}$ is found, then φ is unsatisfiable and the algorithm ends here. Otherwise, OBV-BS enters a loop over the bits in the objective function, starting from the leading MSB $\mathsf{obj}[0]$ and ending with the trailing LSB $\mathsf{obj}[n-1]$. If $\mathsf{obj}[i]$ is true in $\mathcal{M}$, then $\mathsf{obj}[i]$ cannot be improved and it is thus added to an initially empty list of of assumptions Γ. After that, the execution flow jumps at the next loop iteration, or stops if the LSB has just been visited. Otherwise, the algorithm checks whether the value of the objective function can be improved by assigning $\mathsf{obj}[i]$ to true. This is done by verifying the satisfiability of φ under assumptions, whereby the assumptions list is given by $\Gamma \cup \mathsf{obj}[i]$. If that's indeed the case and the underlying solver answers with SAT, then $\mathsf{obj}[i]$ is added to the list of assumptions Γ and $\mathcal{M}$ is updated with a new model that is closer to the optimal solution. Otherwise, Γ is extended with $\neg\mathsf{obj}[i]$. In either case, the search jumps at the beginning of the loop with the next bit of the $\mathcal{BV}$ objective being evaluated, if any. At the end of the loop, the optimal solution can be extracted by the model $\mathcal{M}$ that was most recently found, or extrapolated from the list of assumptions Γ.

Similarly to OBV-WA, the performance of OBV-BS can be enhanced by initializing the phase saving value of each $\mathsf{obj}[i]$ to 1 prior to any satisfiability check performed with the underlying SMT solver [NR16].

Definitions 4.3.2 and 4.3.3 with Theorem 4.3.4 suggest a direct extension to the minimization/maximization of *signed* $\mathcal{BV}$ of the OBV-BS algorithm for unsigned $\mathcal{BV}$ in [NR16]: *apply the unsigned-$\mathcal{BV}$ maximization [resp. minimization] algorithm of [NR16] to the objective* $\mathsf{obj}' \stackrel{def}{=} (\mathsf{obj}\ \mathbf{nxor}_n\ attr)$ *[resp.* $\mathsf{obj}' \stackrel{def}{=} (\mathsf{obj}\ \mathbf{xor}_n\ attr)$*] instead than simply to* obj *[resp.* $\overline{\mathsf{obj}}$*].*

Experimental Evaluation. In Section §6.3, we reproduce the experimental evaluation contained in [NR16], and compare the effect of enabling the proposed enhancements on the performance of OPTIMATHSAT.

4.4 OMT ($\mathcal{FP} \cup \mathcal{T}$)

In the following, we describe the OMT($\mathcal{FP} \cup \mathcal{T}$) handling of OPTIMATHSAT described in [TS19], which is based on the OMT($\mathcal{BV} \cup \mathcal{T}$) procedures presented by Nadel et al. in [NR16].

Without any loss of generality, we assume that every non-trivial $\mathcal{FP}$ objective function $f(...)$ is replaced by a $\mathcal{FP}$ variable obj by conjoining "$\mathsf{obj} = f(...)$" to the input formula. Following the same conventions established in Section §2.2.3, we indifferently represent a $\mathcal{FP}$ objective

obj with sort `(_ FP <ebits> <sbits>)` as a vector of n bits $[\mathsf{obj}[0], ..., \mathsf{obj}[n-1]]$, where $n \stackrel{\text{def}}{=} ebits + sbits$, or as a triplet of Bit-Vectors $\langle \textbf{sign}, \textbf{exp}, \textbf{sig} \rangle$ such that **sign** is a $\mathcal{BV}$ of size 1, **exp** is a $\mathcal{BV}$ of size $ebits$ and **sig** is a $\mathcal{BV}$ of size $sbits - 1$. We write $\mathsf{obj}[i]$ to denote the i-th bit of obj, where $\mathsf{obj}[0]$ and $\mathsf{obj}[n-1]$ are the Most Significant Bit (MSB) and the Least Significant Bit (LSB) of obj respectively.[3] We use the symbol μ_k to denote a generic (possibly partial) assignment that assigns at least the k most-significant bits of obj. We use the symbol τ_k to denote an assignment to the k most-significant bits of obj. Given $i < k$, we denote by $\mu_k[i]$ [resp. $\tau_k[i]$] the value in $\{0, 1\}$ assigned to $\mathsf{obj}[i]$ by μ_k [resp. τ_k]. Moreover, we use the expression $[\![\mu_k]\!]_i$ where $i \leq k$ to denote the restriction of μ_k to the i most-significant bits of obj, $\mathsf{obj}[0], ..., \mathsf{obj}[i-1]$. Given a model $\mathcal{M}$ of φ and a variable v, we denote by $\mathcal{M}(v)$ the evaluation of v in $\mathcal{M}$.

We define the *Floating-Point Optimization problem* as follows.

Definition 4.4.1. *(OMT($\mathcal{FP} \cup \mathcal{T}$), OMT($\mathcal{FP}$), $\mathsf{min}_{\mathsf{obj}}$). Let φ be a ground SMT($\mathcal{FP} \cup \mathcal{T}$) formula and* obj *be a $\mathcal{FP}$ variable occurring in φ. We call an Optimization Modulo $\mathcal{FP} \cup \mathcal{T}$ problem, the problem of finding a model $\mathcal{M}$ for φ (if any) whose value of* obj*, denoted with $\mathsf{min}_{\mathsf{obj}}(\varphi)$, is either*

- *minimum with respect to the usual total order relation $\leq$ for $\mathcal{FP}$ numbers, if φ is satisfied by at least one model $\mathcal{M}'$ such that $\mathcal{M}'(\mathsf{obj})$ is not* NAN*,*
- *some binary representation of* NAN*, otherwise.*

We call an Optimization Modulo $\mathcal{FP}$ problem, written OMT($\mathcal{FP}$), an OMT($\mathcal{FP} \cup \mathcal{T}$) problem where $\mathcal{T}$ is the empty theory. (The dual definition where we look for the maximum follows straightforwardly)

Definition 4.4.1 is made complicated by the fact that obj can be NAN. In fact, in the SMT-LIBV2 standard the comparisons $\{\leq, <, \geq, >\}$ between NAN and any other $\mathcal{FP}$ value always evaluated to false because NAN has multiple representations at the binary level (see Table 4.1). Also, requiring the optimal solution to be always different from NAN makes the resulting OMT($\mathcal{FP} \cup \mathcal{T}$) problem $\langle \varphi \wedge \neg \mathsf{IsNaN}(\mathsf{obj}), \mathsf{obj} \rangle$ unsatisfiable when φ is satisfied only by models $\mathcal{M}$ such that $\mathcal{M}(\mathsf{obj})$ is NAN. For these reasons, we admit NAN as the optimal solution value for obj if and only if φ is satisfied only by models $\mathcal{M}$ such that $\mathcal{M}(\mathsf{obj})$ is NAN.

In the rest of this section we assume that we have already checked, in sequence, that

i) the input formula φ is satisfiable —by invoking an SMT($\mathcal{FP} \cup \mathcal{T}$) solver on φ. If the solver returns UNSAT, then there is no need to proceed;

ii) φ is satisfied by at least one model $\mathcal{M}'$ such that $\mathcal{M}'(\mathsf{obj})$ is not NAN —by invoking an SMT($\mathcal{FP} \cup \mathcal{T}$) solver on $\varphi \wedge \neg\mathsf{IsNaN}(\mathsf{obj})$ if the model $\mathcal{M}$ returned by the previous SMT call is such that $\mathcal{M}(\mathsf{obj})$ is NAN. If the solver returns UNSAT, then we conclude that the minimum is NAN.

After that, we can safely focus our investigation on the restricted OMT($\mathcal{FP} \cup \mathcal{T}$) problem $\langle \varphi_{\mathsf{noNaN}}, \mathsf{obj} \rangle$, where $\varphi_{\mathsf{noNaN}} \stackrel{\text{def}}{=} \varphi \wedge \neg\mathsf{IsNaN}(\mathsf{obj})$, knowing it is satisfiable.

In Section §4.3, we have introduced the concept of a $\mathcal{BV}$ objective attractor, and we have shown how this can be used to drive the optimization search towards the optimum value, when minimizing or maximizing a *signed* or an *unsigned* $\mathcal{BV}$ goal. However, in the case of floating-point optimization, it is not possible to statically determine the attractor value, before the search is even started. This is due to the more complex representation of $\mathcal{FP}$ variables, that uses three separate Bit-Vectors (i.e. sign, exponent and significand), and the presence of various classes of special values (i.e. zeros, infinity, NAN), that make Definition 4.3.2 ambiguous for $\mathcal{FP}$ optimization. We illustrate this problem with the following example.

Example 4.4.1. *Let $\langle \varphi_{\mathsf{noNaN}}, \mathsf{obj} \rangle$ be an OMT($\mathcal{FP} \cup \mathcal{T}$) problem where* obj *is a $\mathcal{FP}$ objective, of sort* `(_ FP 3 5)`*, to be minimized. To make our explanation easier to follow, we show in Table 4.1 a short list of sample values for an $\mathcal{FP}$ variable of the same sort as* obj*. Each $\mathcal{FP}$ value is represented as a triplet of Bit-Vectors $\langle \mathbf{sign}, \mathbf{exp}, \mathbf{sig} \rangle$ —following the* SMT-LIBv2 *conventions described in Section §2.2.3— and also in decimal notation.*

From Table 4.1, we immediately notice that the binary representation of both the exponent and the significant of a floating-point number grows in opposite directions in the positive and in the negative domains. In addition, by sorting the values according to their binary representation, we observe that $-\infty$ [resp. $+\infty$] is not the smallest [resp. greatest] representable $\mathcal{FP}$ value in the negative [resp. positive] domain. In fact, both extreme ends of the table are occupied by NAN*, that has multiple binary representations.*

In what follows, we temporarily disregard the effects of unit-propagation, that might assign some (or all) bits of obj *as a result of some constraints in φ_{noNaN}, and pick some values as candidate attractors for an $\mathcal{FP}$ goal to be minimized.*

Suppose that the attractor is chosen to be equal to the value $-\infty$ listed at row 9 in Table 4.1, which is the smallest $\mathcal{FP}$ value with respect to the total order relation $\leq$ for $\mathcal{FP}$ numbers. Assume that the optimal value of the $\mathcal{FP}$ goal is the subnormal $\mathcal{FP}$ value `(fp #b1 #b000 #b1111)` *(i.e. $\frac{-15}{64}$). Then, it can be seen that after both the sign and the exponent bits have been decided to be equal* `#b1` *and* `#b000` *respectively, the remaining bits of the attractor pull the search in the wrong direction, that is, towards -0.*

	sign	exp	sig	value
1	#b0	#b111	#b1111	NAN
	...	...	...	NAN
2	#b0	#b111	#b0000	$+\infty$
3	#b0	#b110	#b1111	$\frac{31}{2}$
	...	...	...	...
4	#b0	#b000	#b0001	$\frac{1}{64}$
5	#b0	#b000	#b0000	$+0$
6	#b1	#b000	#b0000	-0
7	#b1	#b000	#b0001	$-\frac{1}{64}$
	...	...	...	...
8	#b1	#b110	#b1111	$-\frac{31}{2}$
9	#b1	#b111	#b0000	$-\infty$
	...	...	...	NAN
10	#b1	#b111	#b1111	NAN

Table 4.1: Sample values for a $\mathcal{FP}$ variable with sort `(_ FP 3 5)`.

Selecting a different $\mathcal{FP}$ value as candidate attractor does not really solve the problem; rather, it results in a different set of issues.

For instance, an attractor equal to the NAN *value listed at row* 10 *in Table 4.1, which is the smallest representable $\mathcal{FP}$ value according to the binary ordering, would solve the problem for the previous case in which the optimum $\mathcal{FP}$ value is* `(fp #b1 #b000 #b1111)`*. However, this attractor would remain an unsuitable choice for an OMT($\mathcal{FP} \cup \mathcal{T}$) instance where the $\mathcal{FP}$ goal is forced to be positive, because after the sign bit of the objective function has been decided to be equal* `#b0` *the remaining bits of the attractor drive the search in the wrong direction, that is, towards $+\infty$.* ◇

Since there is no statically determined $\mathcal{FP}$ value that can be used as an attractor when dealing with floating-point optimization, we introduce the new concept of *dynamic attractor*.

Definition 4.4.2. *(Dynamic Attractor.) Let $\langle \varphi_{\mathsf{noNaN}}, \mathsf{obj} \rangle$ be a restricted OMT($\mathcal{FP} \cup \mathcal{T}$) problem, where $\varphi_{\mathsf{noNaN}} \stackrel{def}{=} \varphi \wedge \neg \mathsf{IsNaN}(\mathsf{obj})$ is a satisfiable SMT($\mathcal{FP} \cup \mathcal{T}$) formula and* obj *is a $\mathcal{FP}$ objective to be minimized [resp. maximized]. Let $k \in [0..n]$ and τ_k be an assignment to the k most-significant bits of* obj*.*

Then, we say that an $\mathcal{FP}$-value $attr_{\tau_k}$ for obj *is a* dynamic attractor for obj with respect to τ_k *if and only if it is the smallest [resp. largest] $\mathcal{FP}$ value different from*

NAN *such that the k most-significant bits of $attr_{\tau_k}$ have the same value of the k most-significant bits of* obj *in τ_k. We call* vector of attractor equalities *the vector A_{τ_k} such that* $A_{\tau_k}[i] \stackrel{def}{=} (\mathsf{obj}[i] = attr_{\tau_k}[i])$, $i \in [0..n-1]$.

With a small abuse of notation, and when this does not cause ambiguities, we sometimes use an dynamic attractor equality $A_{\tau_k}[i] \stackrel{\text{def}}{=} (\mathsf{obj}[i] = attr_{\tau_k}[i])$ to denote the single-bit assignment $\mathsf{obj}[i] := attr_{\tau_k}[i]$ and its negation $\neg A_{\tau_k}[i]$ to denote the assignment to the complement value $\mathsf{obj}[i] := \overline{attr_{\tau_k}[i]}$.

The following fact derives from the above definitions and the properties of IEEE 754-2008 standard representation adopted by SMT-LIBv2 standard for $\mathcal{FP}$.

Lemma 4.4.3. Let $\langle \varphi_{\mathsf{noNaN}}, \mathsf{obj} \rangle$ be a restricted minimization [resp. maximization] OMT($\mathcal{FP} \cup \mathcal{T}$) problem, let τ_k be an assignment to $\mathsf{obj}[0]...\mathsf{obj}[k-1]$ and $attr_{\tau_k}$ be its corresponding dynamic attractor, for some $k \in [0..n-1]$. Let $\tau_{k+1} \stackrel{\text{def}}{=} \tau_k \cup \{\mathsf{obj}[k] := attr_{\tau_k}[k]\}$ and $\tau'_{k+1} \stackrel{\text{def}}{=} \tau_k \cup \{\mathsf{obj}[k] := \overline{attr_{\tau_k}[k]}\}$, and let $\mathcal{M}$, $\mathcal{M}'$ two models for φ_{noNaN} that extend τ_{k+1} and τ'_{k+1} respectively.

Then $\mathcal{M}(\mathsf{obj}) \leq \mathcal{M}'(\mathsf{obj})$ [resp. $\mathcal{M}(\mathsf{obj}) \geq \mathcal{M}'(\mathsf{obj})$].

Proof. (We prove the case of minimization, since that of maximization is dual with respect to the value of the sign bit.) We distinguish three cases based on the value of k.

Case $k = 0$ **(sign bit)**. Then $attr_{\tau_0}[0] = 1$, $\tau_1 = \{\mathsf{obj}[0] = 1\}$ and $\tau'_1 = \{\mathsf{obj}[0] = 0\}$, where $\mathsf{obj}[0]$ is the MSB of obj and represents the sign of the floating-point value. Then obj is smaller or equal zero in every model $\mathcal{M}$ and larger or equal zero in every model $\mathcal{M}'$ of φ_{noNaN}, so that $\mathcal{M}(\mathsf{obj}) \leq \mathcal{M}'(\mathsf{obj})$ is verified.

Case $k \in [1..ebits]$ **(exponent bits)**, where $ebits$ is the number of bits in the exponent of obj. Then, $attr_{\tau_k}[k]$ is 1 if $\tau_k[0] = 1$ and 0 otherwise.

In the first case, obj can only be negative in both $\mathcal{M}$ and $\mathcal{M}'$. More precisely, $\mathcal{M}(\mathsf{obj})$ can be either $-\infty$ or a normal negative value, whereas $\mathcal{M}'(\mathsf{obj})$ can be either a normal or a subnormal negative value. Hereafter, we consider only the case in which both have a normal negative value, because the case in which $\mathcal{M}(\mathsf{obj}) = -\infty$ or $\mathcal{M}'(\mathsf{obj})$ is subnormal are both trivial, given that the absolute value of any subnormal $\mathcal{FP}$ number is smaller than the absolute value of any normal $\mathcal{FP}$ number. Furthermore, we disregard the significand bits in $\mathcal{M}$ and $\mathcal{M}'$ because their contribution to the value of obj is always less significant than that of the bits in the exponent. Given these premises, the exponent value of obj in every possible $\mathcal{M}$ is larger than the exponent of obj in every possible $\mathcal{M}'$ by a value equal to $2^{ebits-k}$ and therefore, given that both $\mathcal{M}(\mathsf{obj})$ and $\mathcal{M}'(\mathsf{obj})$ are negative, $\mathcal{M}(\mathsf{obj}) \leq \mathcal{M}'(\mathsf{obj})$.

The case in which $\tau_k[0] = 0$, that is when obj can only be positive in both $\mathcal{M}$ and $\mathcal{M}'$, is dual.

Case $k > ebits$ **(significand bits)**. Then there are three subcases.

If for every $i \in [1..ebits]$ the value of $\tau_k[i]$ is equal 1, then the only possible value of $\mathcal{M}(\mathsf{obj})$ for every possible $\mathcal{M}$ is $+\infty$, and therefore $attr_{\tau_k}[k] = 0$. On the other hand, there exists no possible model $\mathcal{M}'$ of φ_{noNaN}, because the assignment $\mathsf{obj}[k] = 1$ would imply obj being equal to NAN, so the statement $\mathcal{M}(\mathsf{obj}) \leq \mathcal{M}'(\mathsf{obj})$ is vacuously true.

If instead there is some $i \in [1..ebits]$ such that $\tau_k[i] = 0$, then $attr_{\tau_k}[k]$ is 1 if $\tau_k[0] = 1$ (i.e. obj is negative) and 0 otherwise (i.e. obj is positive). In both cases, we can disregard the exponent bits in $\mathcal{M}$ and $\mathcal{M}'$ because their contribution to the value of obj is the same in either model. For the same reasons, since $\mathcal{M}(\mathsf{obj})$ and $\mathcal{M}'(\mathsf{obj})$ can only be either both normal or both subnormal, we can ignore the contribution of the leading hidden bit and focus on the bits of the significand.

When $\tau_k[0] = 1$ and obj must be negative, the decimal value of the significand in $\mathcal{M}$ is larger than the decimal value of every possible significand in $\mathcal{M}'$ by exactly $2^{-(k-ebits)}$. Given that both $\mathcal{M}(\mathsf{obj})$ and $\mathcal{M}'(\mathsf{obj})$ are negative, we have that $\mathcal{M}(\mathsf{obj}) \leq \mathcal{M}'(\mathsf{obj})$.

The case in which $\tau_k[0] = 0$, that is when obj can only be positive in both $\mathcal{M}$ and $\mathcal{M}'$, is dual. □

Lemma 4.4.3 states that, given the current assignment τ_k to the k most-significant bits of obj, $\mathsf{obj}[k] = attr_{\tau_k}[k]$ is always the best extension of τ_k to the next bit (when it does not conflict with φ_{noNaN}). A dynamic attractor $attr_{\tau_k}$ can thus be used by the optimization search to guide the assignment of the $k+1$-th bit of obj towards the direction of maximum gain that is allowed by τ_k, so that to obtain the "best" extension τ_{k+1} of τ_k. Once the (new) assignment τ_{k+1} is found, the OMT solver can compute the dynamic attractor $attr_{\tau_{k+1}}$ for obj with respect to τ_{k+1} and then use it to assign the $k+2$-th bit of obj, and so on.

Let $\langle \varphi_{\mathsf{noNaN}}, \mathsf{obj} \rangle$ be an OMT($\mathcal{FP} \cup \mathcal{T}$) instance, such that obj is a $\mathcal{FP}$ variable of n bits, and τ_0 be an initially empty assignment. If at each step of the optimization search the assignment of the k-th bit of obj is guided by the dynamic attractor for obj with respect to τ_k, then the corresponding sequence of n dynamic attractors (of increasing order k) is unique and depends exclusively on φ_{noNaN}. Intuitively, this is the case because the (current) dynamic attractor always points in the direction of maximum gain. We illustrate this in the following example.

Example 4.4.2. *Let* $\langle \varphi_{\mathsf{noNaN}}, \mathsf{obj} \rangle$ *be an OMT*($\mathcal{FP} \cup \mathcal{T}$) *problem where* obj *is a* $\mathcal{FP}$ *objective, of sort* `(_ FP 3 5)`*, to be minimized, as in Example 4.4.1. At the beginning of the search, nothing is known about the structure of the solution. Therefore,* $\tau_0 = \emptyset$ *and, since* obj *is*

being minimized, the dynamic attractor *for* obj *with respect to* τ_0 *(i.e.* $attr_{\tau_0}$*) is equal to* `(fp #b1 #b111 #b0000)` *(i.e.* $-\infty$*), that gives a preference to any feasible value of* obj *in the negative domain.*

If at some point of the optimization search we discover that the domain of the objective function can only be positive, so that the first bit of obj *is permanently set to* 0 *in* τ_1*, then the new dynamic attractor for* obj *with respect to* τ_1 *(i.e.* $attr_{\tau_1}$*) is equal to* `(fp #b0 #b000 #b0000)` *(i.e.* $+0$*).*

Furthermore, if later on we also find out that at least one bit in the exponent of obj *can be assigned to* 0 *in a feasible solution of the problem that extends* τ_i*, for some* i*, then we can remove* $+\infty$ *from the optimization search interval.* ◇

Definition 4.4.4. *(Attractor Trajectory* $\mathcal{A}_\varphi$*). Consider the restricted OMT*$(\mathcal{FP} \cup \mathcal{T})$ *problem* $\langle \varphi_{\mathsf{noNaN}}, \mathsf{obj} \rangle$ *such that* $\varphi_{\mathsf{noNaN}} \stackrel{def}{=} \varphi \wedge \neg \mathsf{IsNaN}(\mathsf{obj})$ *as in Definition 4.4.2, a triplet of inductively-defined sequences* $\langle \{\tau_0, \tau_1, ..., \tau_n\}, \{attr_{\tau_0}, attr_{\tau_1}, ..., attr_{\tau_n}\}, \{A_{\tau_0}, A_{\tau_1}, ..., A_{\tau_n}\} \rangle$ *—where each* τ_k *is an assignment to the first* k *most-significant bits of* obj *such that* $\tau_k \subset \tau_{k+1}$*,* $attr_{\tau_k}$ *is its corresponding dynamic attractor and* A_{τ_k} *is its corresponding* vector of attractor equalities*— so that, for every* $k \in [0..n-1]$*:*

(i) $\tau_{k+1}[k] = \overline{attr_{\tau_k}[k]}$ *if* $\varphi_{\mathsf{noNaN}} \wedge \tau_k \wedge A_{\tau_k}[k]$ *is unsatisfiable,*

(ii) $\tau_{k+1}[k] = attr_{\tau_k}[k]$ *otherwise.*

Then we define the attractor trajectory $\mathcal{A}_\varphi$ *as the vector* $[A_{\tau_0}[0], ..., A_{\tau_{n-1}}[n-1]]$.

The attractor trajectory $\mathcal{A}_\varphi$ contains those attractor equalities ($\mathsf{obj}[k] = attr_{\tau_k}[k]$) that are of critical importance for the decisions taken by the optimization search. Intuitively, this is the case because the value of the k-th bit of obj (i.e. $\mathsf{obj}[k]$) is still undecided in τ_k.

Example 4.4.3. *Let* $\langle \varphi_{\mathsf{noNaN}}, \mathsf{obj} \rangle$ *be a restricted OMT*$(\mathcal{FP} \cup \mathcal{T})$ *problem where* obj *is a* $\mathcal{FP}$ *objective, of sort* `(_ FP 3 5)`*, to be minimized, as in Example 4.4.1. We consider the case in which the input formula* φ_{noNaN} *requires* obj *to be larger or equal* $29/2$ *and it does not impose any other constraint on the value of* obj*. Given the sequence of (partial) assignments* $\tau_0, ..., \tau_8$ *in Figure 4.5, the corresponding list of dynamic attractors and the corresponding vectors of attractor equalities, then the attractor trajectory* $\mathcal{A}_\varphi$ *is equal to the vector* $[\mathsf{obj}[0] = 1, \mathsf{obj}[1] = 0, \mathsf{obj}[2] = 0, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 0, \mathsf{obj}[5] = 0, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0]$. ◇

Lemma 4.4.5. Consider $\langle \varphi_{\mathsf{noNaN}}, \mathsf{obj} \rangle$, $\tau_0, ..., \tau_n$, $attr_{\tau_0}, ..., attr_{\tau_n}$, $A_{\tau_0}, ..., A_{\tau_n}$, and $\mathcal{A}_\varphi$

$\tau_0 = \emptyset$ $\quad attr_{\tau_0} = \texttt{(fp \#b1 \#b111 \#b0000)} = [\underline{1}.111.1111]$ [i.e. $-\infty$] $\implies$ UNSAT

$\tau_1 = \tau_0 \cup \{\mathsf{obj}[0] = 0\}$ $\quad attr_{\tau_1} = \texttt{(fp \#b0 \#b000 \#b0000)} = [0.\underline{0}00.0000]$ [i.e. +0] $\implies$ UNSAT

$\tau_2 = \tau_1 \cup \{\mathsf{obj}[1] = 1\}$ $\quad attr_{\tau_2} = \texttt{(fp \#b0 \#b100 \#b0000)} = [0.1\underline{0}0.0000]$ [i.e. +2] $\implies$ UNSAT

$\tau_3 = \tau_2 \cup \{\mathsf{obj}[2] = 1\}$ $\quad attr_{\tau_3} = \texttt{(fp \#b0 \#b110 \#b0000)} = [0.11\underline{0}.0000]$ [i.e. +8] $\implies$ SAT

$\tau_4 = \tau_3 \cup \{\mathsf{obj}[3] = 0\}$ $\quad attr_{\tau_4} = \texttt{(fp \#b0 \#b110 \#b0000)} = [0.110.\underline{0}000]$ [″ ″] $\implies$ UNSAT

$\tau_5 = \tau_4 \cup \{\mathsf{obj}[4] = 1\}$ $\quad attr_{\tau_5} = \texttt{(fp \#b0 \#b110 \#b1000)} = [0.110.1\underline{0}00]$ [i.e. +12] $\implies$ UNSAT

$\tau_6 = \tau_5 \cup \{\mathsf{obj}[5] = 1\}$ $\quad attr_{\tau_6} = \texttt{(fp \#b0 \#b110 \#b1100)} = [0.110.11\underline{0}0]$ [i.e. +14] $\implies$ SAT

$\tau_7 = \tau_6 \cup \{\mathsf{obj}[6] = 0\}$ $\quad attr_{\tau_7} = \texttt{(fp \#b0 \#b110 \#b1100)} = [0.110.1100]$ [″ ″] $\implies$ UNSAT

$\tau_8 = \tau_7 \cup \{\mathsf{obj}[7] = 1\}$ $\quad attr_{\tau_8} = \texttt{(fp \#b0 \#b110 \#b1101)} = [0.110.1101]$ [i.e. $^{29}/_{2}$]

$A_{\tau_0} = [\underline{\mathsf{obj}[0] = 1}, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \mathsf{obj}[3] = 1, \mathsf{obj}[4] = 0, \mathsf{obj}[5] = 0, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0]$

$A_{\tau_1} = [\mathsf{obj}[0] = 0, \underline{\mathsf{obj}[1] = 0}, \mathsf{obj}[2] = 0, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 0, \mathsf{obj}[5] = 0, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0]$

$A_{\tau_2} = [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \underline{\mathsf{obj}[2] = 0}, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 0, \mathsf{obj}[5] = 0, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0]$

$A_{\tau_3} = [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \underline{\mathsf{obj}[3] = 0}, \mathsf{obj}[4] = 0, \mathsf{obj}[5] = 0, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0]$

$A_{\tau_4} = [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \mathsf{obj}[3] = 0, \underline{\mathsf{obj}[4] = 0}, \mathsf{obj}[5] = 0, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0]$

$A_{\tau_5} = [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 1, \underline{\mathsf{obj}[5] = 0}, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 0]$

$A_{\tau_6} = [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 1, \mathsf{obj}[5] = 1, \underline{\mathsf{obj}[6] = 0}, \mathsf{obj}[7] = 0]$

$A_{\tau_7} = [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 1, \mathsf{obj}[5] = 1, \mathsf{obj}[6] = 0, \underline{\mathsf{obj}[7] = 0}]$

$A_{\tau_8} = [\mathsf{obj}[0] = 0, \mathsf{obj}[1] = 1, \mathsf{obj}[2] = 1, \mathsf{obj}[3] = 0, \mathsf{obj}[4] = 1, \mathsf{obj}[5] = 1, \mathsf{obj}[6] = 0, \mathsf{obj}[7] = 1]$

Figure 4.5: An example of $\mathcal{FP}$ optimization using the dynamic attractor. ("$\implies$ SAT/UNSAT" denotes the satisfiability of $\varphi_{\mathsf{noNaN}} \wedge \tau_k \wedge A_{\tau_k}[k]$, the symbols "″ ″" stand for "the same as above". For ease of illustration, we have underlined the critical bit $attr_{\tau_k}[k]$ in the attractors and each attractor equality of the attractor trajectory $\mathcal{A}_\varphi$ inside the vectors of attractor equalities.)

as in Definition 4.4.4. Then τ_n lexicographically maximizes $\mathcal{A}_\varphi$ with respect to φ_{noNaN}.

Proof. By Definition 4.4.4, we have that, for each $k \in [0..n-1]$,

(i) $\tau_{k+1}[k] = \overline{attr_{\tau_k}[k]}$ if $\varphi_{\mathsf{noNaN}} \wedge \tau_k \wedge A_{\tau_k}[k]$ is unsatisfiable,

(ii) $\tau_{k+1}[k] = attr_{\tau_k}[k]$ otherwise.

By construction, $\tau_k = [\![\tau_n]\!]_k$. Therefore, we can replace τ_k with $[\![\tau_n]\!]_k$ so that

(i) $[\![\tau_n]\!]_{k+1}[k] = \overline{attr_{[\![\tau_n]\!]_k}[k]}$ if $\varphi_{\mathsf{noNaN}} \wedge [\![\tau_n]\!]_k \wedge A_{[\![\tau_n]\!]_k}[k]$ is unsatisfiable,

(ii) $[\![\tau_n]\!]_{k+1}[k] = attr_{[\![\tau_n]\!]_k}[k]$ otherwise.

We notice the following facts. For each $k \in [0..n-1]$, $[\![\tau_n]\!]_k \subset \tau_n$. Furthermore, for each $k \in [0..n-1]$, $\mathcal{A}_\varphi[k] = A_{[\![\tau_n]\!]_k}[k]$ because $\mathcal{A}_\varphi[k] = A_{\tau_k}[k]$ by the definition of attractor trajectory, and $A_{\tau_k}[k] = A_{[\![\tau_n]\!]_k}[k]$ by the equality $\tau_k = [\![\tau_n]\!]_k$. Thus, we can replace $[\![\tau_n]\!]_{k+1}$ with τ_n and $A_{[\![\tau_n]\!]_k}[k]$ with $\mathcal{A}_\varphi[k]$, as follows. For each $k \in [0..n-1]$,

(i) $\tau_n[k] = \overline{attr_{\tau_n}[k]}$ if $\varphi_{\mathsf{noNaN}} \wedge [\![\tau_n]\!]_k \wedge \mathcal{A}_\varphi[k]$ is unsatisfiable,

(ii) $\tau_n[k] = attr_{\tau_n}[k]$ otherwise.

Hence, τ_n lexicographically maximizes $\mathcal{A}_\varphi$ with respect to φ_{noNaN}. □

Finally, we make the following two observations. The first is that the sequence $\tau_0, \tau_1, ..., \tau_n$ in Definition 4.4.4 can be iteratively constructed using its list of requirements, for instance, by means of a sequence of incremental calls to an SMT solver. The second, more important, observation is that τ_n corresponds to the assignment of values that makes obj optimal in φ_{noNaN}.

Using the above definitions, we show that the following fact holds.

Theorem 4.4.6. Let $\langle \varphi_{\mathsf{noNaN}}, \mathsf{obj} \rangle$, $\tau_0, ..., \tau_n$, $attr_{\tau_0}, ..., attr_{\tau_n}$, $A_{\tau_0}, ..., A_{\tau_n}$, and $\mathcal{A}_\varphi$ be as in Definition 4.4.4. Then, any model $\mathcal{M}$ of φ_{noNaN} that lexicographically maximizes the attractor trajectory $\mathcal{A}_\varphi$ is an optimal solution for the OMT($\mathcal{FP} \cup \mathcal{T}$) problem $\langle \varphi_{\mathsf{noNaN}}, \mathsf{obj} \rangle$.

Proof. (We prove the case of minimization, since that of maximization is dual.)
By Lemma 4.4.5 we have that τ_n lexicographically maximize $\mathcal{A}_\varphi$. Let $\mathcal{M}$ be a model of φ_{noNaN} that lexicographically maximizes $\mathcal{A}_\varphi$, and let μ be its restriction to obj. Since both τ_n and $\mathcal{M}$ lexicographically maximize $\mathcal{A}_\varphi$, for the uniqueness of τ_n, we immediately notice that $\mu = \tau_n$, so that $\tau_k = [\![\mu]\!]_k$ for each $k \in [0..n]$ and μ lexicographically maximize $\mathcal{A}_\varphi$.

By definition, $\mathcal{M}$ is an optimal solution for $\langle \varphi_{\mathsf{noNaN}}, \mathsf{obj} \rangle$ if and only if there exists no other model $\mathcal{M}'$ for it such that $\mathcal{M}'(\mathsf{obj}) < \mathcal{M}(\mathsf{obj})$. Hence, we show by contradiction that no such $\mathcal{M}'$ can exist.

Assume (for the sake of contradiction), that there exists a model $\mathcal{M}'$ for φ_{noNaN}, such that $\mathcal{M}'(\mathsf{obj}) < \mathcal{M}(\mathsf{obj})$, and let μ' be the restriction of $\mathcal{M}'$ to obj. Then there must be at least one index i for which $\mu[i] \neq \mu'[i]$. Let m be the smallest such index. Recalling that $\tau_m = [\![\mu]\!]_m$ and $\tau_{m+1} = [\![\mu]\!]_{m+1}$, we set $\tau'_{m+1} \stackrel{\text{def}}{=} [\![\mu']\!]_{m+1}$. Then, $\tau_m \subset \tau_{m+1}$, $\tau_m \subset \tau'_{m+1}$, $\tau_{m+1} \neq \tau'_{m+1}$. In particular, $\tau_{m+1}[m] = \overline{\tau'_{m+1}[m]}$ and therefore $\tau_{m+1}[m] = attr_{\tau_m}[m]$ if $\tau'_{m+1}[m] = \overline{attr_{\tau_m}[m]}$, and vice versa.

Then, we distinguish two cases.

In the first case, $\tau_{m+1}[m] = \overline{attr_{\tau_m}[m]}$ and $\tau'_{m+1}[m] = attr_{\tau_m}[m]$. From $\tau_{m+1}[m] =$

$\overline{attr_{\tau_m}[m]}$ and the fact that μ lexicographically maximizes $\mathcal{A}_\varphi$, we derive that $\varphi_{\mathsf{noNaN}} \wedge \tau_m \wedge \mathcal{A}_\varphi[m]$ is unsatisfiable, where $\mathcal{A}_\varphi[m] \stackrel{\text{def}}{=} (\mathsf{obj}[m] = attr_{\tau_m}[m])$. Since $\tau_m \subset \tau'_{m+1} \subseteq \mu'$ and $\tau'_{m+1}[m] = attr_{\tau_m}[m]$, we conclude that $\varphi_{\mathsf{noNaN}} \wedge \mu' \models \bot$, so that $\mathcal{M}'$ cannot be a model of φ_{noNaN}, contradicting the initial assumption.

In the second case, $\tau_{m+1}[m] = attr_{\tau_m}[m]$ and $\tau_{m+1}[m] = \overline{attr_{\tau_m}[m]}$. Therefore, by Lemma 4.4.3, for every pair of models $\mathcal{M}_1$, $\mathcal{M}_2$ for φ_{noNaN} that extend respectively τ_{m+1} and τ'_{m+1} we have that $\mathcal{M}_1(\mathsf{obj}) \leq \mathcal{M}_2(\mathsf{obj})$. Since $\tau_{m+1} = [\![\mu]\!]_{m+1}$ and $\tau'_{m+1} = [\![\mu']\!]_{m+1}$, it follows that $\mathcal{M}'(\mathsf{obj}) \not\leq \mathcal{M}(\mathsf{obj})$, contradicting the initial assumption. □

In this thesis, we consider two approaches for dealing with OMT($\mathcal{FP} \cup \mathcal{T}$): a basic OMT-based search, using the inline OMT schema presented in §2.3.1, and *Floating-Point Optimization with Binary Search* (OFP-BS), a new engine inspired to the OBV-BS algorithm for Bit-Vector optimization presented in [NR16].

4.4.1 OMT-based Search

The OMT-based approach for OMT($\mathcal{FP} \cup \mathcal{T}$) adapts the linear-, binary- and adaptive-search schemata for OMT($\mathcal{LRA} \cup \mathcal{T}$) —described in §2.3.1— to deal with $\mathcal{FP}$ objectives.

In the basic linear-search schema, the optimization search is advanced by means of a sequence of linear cuts, each of which forces the OMT solver to look for a new model $\mathcal{M}'$ that improves the value of obj with respect to the most recent model $\mathcal{M}$. In the binary-search schema, instead, the OMT solver learns an incremental sequence of pivoting cuts that bisect the current domain of the objective function. Hereafter, we succinctly describe the binary-search schema for OMT($\mathcal{FP} \cup \mathcal{T}$), that requires minimal variations with respect to the binary-search schema for OMT($\mathcal{LRA} \cup \mathcal{T}$) in Section §2.3.1. At the beginning of the optimization search and following each update of the lower (lb) and upper (ub) bounds of obj, the OMT solver computes a (new) pivoting value as follows:

$$\mathsf{pivot} \stackrel{\text{def}}{=} \mathtt{floor}(\rho \cdot \mathsf{ub} + (1 - \rho) \cdot \mathsf{lb})$$

where ρ is contained in the interval $[0, 1[$ (e.g. $\frac{1}{2}$). When minimizing [resp. maximizing], the pivot must be checked to not lie outside the range $]\mathsf{lb}, \mathsf{ub}]$ [resp. $[\mathsf{lb}, \mathsf{ub}[$] due to the side effects of rounding operations. If that is the case, a regular linear search step should be performed instead, as learning the pivoting cut $(\mathsf{obj} < \mathsf{pivot})$ [resp. $(\mathsf{obj} > \mathsf{pivot})$] is not guaranteed to advance the optimization search (e.g. when pivot is equal lb). Alternatively, instead of validating the pivot, a linear search step could simply be guaranteed to be performed infinitely often. If the pivoting cut is satisfiable, the upper bound of obj is updated with the value of obj in the corresponding

model $\mathcal{M}$. Otherwise, the lower bound is made equal to pivot. The algorithm terminates when the search interval $[\mathsf{lb}, \mathsf{ub}[$ becomes empty.

In general, it is reasonable to expect the binary-search schema to converge towards the optimal solution faster than the linear-search schema, because the feasible domain of a $\mathcal{FP}$ goal can be comprised by an exponentially large number of values (with respect to the bit width of the cost function).

In either schema, whenever the optimization engine encounters for the first time a solution such that $\mathsf{obj} = \text{NAN}$, the OMT solver learns a unit clause of the form $\neg(\text{ISNAN}(\mathsf{obj}))$ to look for an optimal solution different from NAN (if any).

Differently from the case of OMT($\mathcal{LRA} \cup \mathcal{T}$) described in Section §2.3.1, when dealing with $\mathcal{FP}$ objectives it is not necessary to implement a specialized optimization procedure within the $\mathcal{FP}$-Solver to guarantee the termination of the optimization search. This is nice because such procedure is actually not available when Floating-Point terms are bit-blasted into Bit-Vectors *eagerly*, or when the ACDCL $\mathcal{FP}$-Solver is used, because by the time the optimization procedure is called the domain interval of any $\mathcal{FP}$ term contains a singleton value. In contrast, such a minimization procedure could be envisaged when the OMT solver uses a *lazy* $\mathcal{FP}$-Solver as back-end to speed up the convergence towards the optimal solution[12].

Enhancements. Assume the choice of an arbitrary $\mathcal{FP}$-value $attr = [attr[0], ..., attr[n-1]]$ as static attractor for the $\mathcal{FP}$ goal $\mathsf{obj} = [\mathsf{obj}[0], ..., \mathsf{obj}[n-1]]$, the corresponding vector of attractor equalities is $[\mathsf{obj}[0] = attr[0], ..., \mathsf{obj}[n-1] = attr[n-1]]$. For each attractor equality of the form $(\mathsf{obj}[i] = attr[i])$, we introduce a fresh Boolean decision variable A_i such that $A_i \leftrightarrow (\mathsf{obj}[i] = attr[i])$. Then, (a combination of) the following techniques can be used to optionally enhance either OMT-based search schema, similarly to the case of OMT($\mathcal{BV}$) described in [NR16].

- **branching preference**: starting from the MSB and down to the LSB, each A_i is marked as a preferred variable for branching. This has the benefit of causing the highest possible backjump when any conflict involving a unit clause in the form $(\mathsf{obj} < \mathsf{lb})$ (resp. $(\mathsf{obj} > \mathsf{ub})$) is encountered while minimizing (resp. maximizing) the objective function.
- **polarity initialization**: the phase-saving value of each A_i is initialized to true at the beginning of the search, so that the first time the SAT engine encounters A_i as a decision variable it tries to assign true first. As a result, the solver prefers looking for candidate values of obj that are closer to the target $attr$.

[12]Currently, there is no such specialized optimization procedure embedded within the *lazy* $\mathcal{FP}$-Solver of OPTIMATHSAT, so we will not describe this approach any further.

In the lucky case, using either (or both) of these techniques can enhance the performance of the OMT solver, by pulling the optimization search in the right direction. In the unlucky case, using either (or both) of these techniques can have no effect, or even hinder the overall performance. For instance, in the case of the linear-search optimization schema, enabling both options with the wrong choice of attractor value can force the OMT solver to start the search from the furthest possible point from the optional solution, and thus enumerate an exponential number of intermediate solutions. Naturally, the OMT-based optimization search algorithm is still guaranteed to terminate even in the worst-case scenario, but the unpredictable performance makes using either technique a generally unsuitable option in practice.

4.4.2 OFP-BS

The *Floating-Point Optimization with Binary Search* algorithm is a new engine for OMT($\mathcal{FP} \cup \mathcal{T}$) that is inspired by the OBV-BS algorithm for OMT($\mathcal{BV}$) [NR16] and is a direct implementation of Definition 4.4.4 and Theorem 4.4.6.

The optimization search tries to lexicographically maximize an implicit *attractor trajectory* vector $\mathcal{A}_\varphi$, that is incrementally derived from the current value of the dynamic attractor. The raw value of the dynamic attractor's bits drive the optimization search towards the direction of maximum gain at any given point in time, without disrupting any decision that has been already made. The dynamic attractor is incrementally updated along the search, based on the outcome of the previous rounds of the optimization search. At each round, one bit of the objective function is assigned its final value. The first round decides the sign, the next batch of rounds decides the exponent and the remaining rounds decide the fine-grained details of the significand.

The pseudocode of OFP-BS is shown in Figure 4.6. The arguments of the algorithm are the input formula φ and the $\mathcal{FP}$ objective obj, where obj is a $\mathcal{FP}$ variable with $ebits$ bits in the exponent, $sbits - 1$ in the significand and $n \stackrel{\text{def}}{=} ebits + sbits$ bits overall.

The procedure starts by checking whether the input formula φ is satisfiable and immediately terminates otherwise (lines 1-3). If $\mathsf{obj} = \text{NAN}$ in $\mathcal{M}$ then the procedure checks whether there exists a model $\mathcal{M}'$ for $\varphi \wedge \neg\mathsf{IsNaN}(\mathsf{obj})$ (lines 4-5). If this is not the case, the procedure terminates immediately and returns the pair $\langle \text{SAT}, \mathcal{M} \rangle$ (line 7). Otherwise, the model $\mathcal{M}$ is updated with the new model $\mathcal{M}'$, and φ is permanently extended with the constraint $\neg\mathsf{IsNaN}(\mathsf{obj})$ (lines 9-10).

At this point, the procedure initializes the value of the dynamic attractor by invoking an external function UPDATE_DYNAMIC_ATTRACTOR() with the empty assignment τ as parameter, so that the returned value is equal to $-\infty$ when minimizing and $+\infty$ when maximizing (lines 11-12). Then, the execution moves to the core part of the OFP-BS algorithm (lines 15-28), which

function OFP-BS (φ, obj)

1: $\langle res, \mathcal{M} \rangle \coloneqq$ SMT.CHECK_UNDER_ASSUMPTIONS$(\varphi, \emptyset)$
2: **if** $(res ==$ UNSAT$)$ **then**
3: **return** $\langle res, \emptyset \rangle$ // φ is unsatisfiable
4: $\varphi \coloneqq \varphi \wedge \neg$isNaN(obj)
5: **if** $(\mathcal{M}$(obj) $==$ NAN$)$ **then**
6: $\langle res, \mathcal{M}' \rangle \coloneqq$ SMT.CHECK_UNDER_ASSUMPTIONS$(\varphi \wedge \neg$isNaN(obj)$, \emptyset)$
7: **if** $(res ==$ UNSAT$)$ **then**
8: **return** $\langle$SAT$, \mathcal{M} \rangle$ // obj can only be NAN
9: **else**
10: $\mathcal{M} \coloneqq \mathcal{M}'$
11: $\varphi \coloneqq \varphi \wedge \neg$isNaN(obj)
12: $\tau \coloneqq \emptyset$ // from now on, obj cannot be equal NAN
13: $attr_\tau \coloneqq$ UPDATE_DYNAMIC_ATTRACTOR(τ)
14: SMT.SET_BRANCHING_PREFERENCE(obj)
15: SMT.UPDATE_BITS_POLARITY_TO(obj, $attr_\tau$)
16: **for** $i \coloneqq 0$ **up to** $n - 1$ **do**
17: $eq \coloneqq$ (obj$[i] = attr_\tau[i]$) // attractor equality $A_\tau[i]$
18: **if** $(\mathcal{M} \models eq)$ **then**
19: $\tau \coloneqq \tau \cup \{eq\}$
20: **else**
21: SMT.SET_BRANCHING_PREFERENCE(obj)
22: SMT.UPDATE_BITS_POLARITY_TO(obj, $attr_\tau$)
23: $\langle res, \mathcal{M}' \rangle \coloneqq$ SMT.CHECK_UNDER_ASSUMPTIONS$(\varphi, \tau \cup \{eq\})$
24: **if** $(res ==$ SAT$)$ **then**
25: $\tau \coloneqq \tau \cup \{eq\}$
26: $\mathcal{M} \coloneqq \mathcal{M}'$
27: **else**
28: $\tau \coloneqq \tau \cup \{\neg eq\}$
29: $attr_\tau \coloneqq$ UPDATE_DYNAMIC_ATTRACTOR(τ)
30: **return** $\langle$SAT$, \mathcal{M} \rangle$

Figure 4.6: OFP-BS Algorithm for Floating-Point optimization.

consists of a loop over the bits of obj, starting from the MSB obj$[0]$ down to the LSB obj$[n-1]$.

Inside this loop, OFP-BS first checks whether the value of obj$[i]$ in $\mathcal{M}$ matches the i-th bit of the (current) dynamic attractor $attr_\tau$. If so, then the i-th bit is already set to its "best" value in $\mathcal{M}$. Thus, the assignment τ is permanently extended with obj$[i] = attr_\tau[i]$ (line 16), and the optimization search moves to the next iteration of the loop. If instead obj$[i] \neq attr_\tau[i]$ in $\mathcal{M}$, we need to verify whether the value of the objective function in $\mathcal{M}$ can be improved by forcing the i-th bit of obj equal to the i-th bit of the dynamic attractor. To do so, we incrementally invoke the underlying SMT solver, this time checking the satisfiability of φ under the list of assumptions $\tau \cup \{\text{obj}[i] = attr_\tau[i]\}$ (line 22). If the SMT solver returns SAT, then the value of the objective function has been successfully improved. Hence, τ is extended with an assignment setting obj$[i]$ equal to $attr_\tau[i]$, and $\mathcal{M}$ is replaced with the new model $\mathcal{M}'$ (lines 23-25). Otherwise, it is not possible to improve the objective function by toggling the value of obj$[i]$, and τ is permanently extended with obj$[i] \neq attr_\tau[i]$ (line 27). At this point, there is a mismatch between the value of the first $i+1$ bits of obj in $\mathcal{M}$, corresponding to the assignment τ, and those of the current dynamic attractor. This mismatch is resolved by calling the function UPDATE_DYNAMIC_ATTRACTOR() with the updated assignment τ as parameter (line 28). In either case, the execution moves to the next iteration of loop.

After exactly n iterations of the loop, the optimization search terminates with the pair $\langle \text{SAT}, \mathcal{M} \rangle$, where $\mathcal{M}$ is the optimum model of the given OMT$(\mathcal{FP} \cup \mathcal{T})$ instance. The OFP-BS algorithm requires at most $n+2$ incremental calls to an underlying SMT$(\mathcal{FP})$ solver. The test in rows 17-18 allows for saving lots of such SMT calls when the current model already assigns obj$[i]$ to its corresponding value in the attractor.

The function UPDATE_DYNAMIC_ATTRACTOR() takes as input τ, a (partial) assignment over the k most-significant bits of obj and, when obj is minimized [13], and it essentially works as follows. If $\tau = \emptyset$, then nothing is known about the solution of the problem, so $-\infty$ is returned. Otherwise, the procedure must compute the smallest $\mathcal{FP}$ value different from NAN (if any) that extends τ. Since $\tau \neq \emptyset$ then we know that the sign of the objective function has been permanently decided in τ. If obj$[0] = 0$ in τ, i.e. obj must be positive, the procedure must return the smallest positive $\mathcal{FP}$ value admitted by τ. Hence, we extend τ with $\bigcup_{i=|\tau|}^{i=n-1} \text{obj}[i] = 0$ and return the corresponding $\mathcal{FP}$ value. If obj$[0] = 1$ in τ, i.e. obj can take negative values, the procedure must return the largest negative $\mathcal{FP}$ value admitted by τ. We first check whether there exists a bit in the exponent of obj that is assigned to 0 in τ. If that is the case, we extend τ with $\bigcup_{i=|\tau|}^{i=n-1} \text{obj}[i] = 1$ and return the corresponding $\mathcal{FP}$ value. Otherwise, the procedure returns the value $-\infty$, that is still a viable extension of τ.

[13] The implementation of UPDATE_DYNAMIC_ATTRACTOR() is dual when obj is maximized.

function UPDATE_DYNAMIC_ATTRACTOR (τ)

```
1:  static attr_τ = −∞                                          // track −∞
2:  if (τ ≠ ∅) then
3:      k := SIZE(τ) − 1
4:      attr_τ[k] = (1 − attr_τ[k])                             // flip current bit
5:      if (τ[0] == 0) then
6:          for i := k + 1 up to n − 1 do
7:              attr_τ[i] = 0                                   // track smallest positive value
8:      else
9:          if (k ≤ ebits) then
10:             for i := k + 1 up to n − 1 do
11:                 attr_τ[i] = 1                               // track largest negative value
12: return attr_τ
```

Figure 4.7: The function UPDATE_DYNAMIC_ATTRACTOR().

Enhancements. The performance of OFP-BS can be improved with one of the following enhancements:

- **branching preference**: the bits of the $\mathcal{FP}$ objective obj are marked inside the SMT solver as preferred variables for branching starting from the MSB down to the LSB (lines 11 and 18). This ensures that conflicts involving the value of the objective function are handled as early as possible, possibly reducing the amount of work that needs to be redone after each backjump.
- **polarity initialization**: prior to any call to the underlying SMT solver, the phase-saving value of each obj$[i]$ is initialized with the value of the corresponding dynamic attractor's bit (rows 12 and 19). This encourages the SMT solver to assign the bits of obj so as to reassemble the bits of the dynamic attractor, thus possibly reducing the number of times that the SMT solver needs to be called.

At the beginning of the search, the OMT solver has no information on the best improving direction to follow. On this regard, we observe that the value of every bit in the dynamic attractor can change after the sign of the objective function has been decided. Furthermore, the value of all the significand's bits in the dynamic attractor can also change during the process of determining the optimal exponent value of the objective function. If the OMT solver applies either enhancement before the correct improving direction is known, this may cause the underlying SMT engine to advance the search starting from a sub-optimal set of initial decisions. This can

be especially the case when both enhancements are enabled at the same time. In order to mitigate this issue, we have designed a variant of our optimization-search approach which does not apply either enhancement on those bits of the objective function for which the best improving direction is not yet known. We call this variant **safe bits restriction**.

Experimental Evaluation. In Section §6.4, we compare the two optimization approaches for Floating-Point objectives on a set of automatically generated benchmarks. Moreover, we compare the effect of enabling the proposed enhancements on the performance of OPTIMATHSAT.

4.5 Incremental OMT

In this section, we define what it means for an OMT solver to be *incremental* and how this feature is realized within OPTIMATHSAT [ST15c]. Incremental OMT is an extension of incremental SMT (see Section §2.2.2), and it is currently available in both Z3 and OPTIMATHSAT.

Definition 4.5.1. *(Incremental OMT). An OMT solver is said to be incremental if it is able to propagate any useful information learned while solving some OMT problem to any future optimization instance. To this aim, an incremental OMT is required to allow for pushing and popping both objectives and clauses from the formula stack at runtime.*

As observed in [ST15c], an OMT solver can be easily made incremental if the underlying SMT solver provides an incremental interface of its own. Luckily, the current trends in both SAT and lazy SMT solving is that of providing an incremental interface, due to the great performance benefit it provides. This is also the case for OPTIMATHSAT, which is based on the SMT solver MATHSAT5. In the following, we illustrate how OPTIMATHSAT exploits MATHSAT5 incremental capabilities to provide its own incremental OMT interface.

Stack Based Incrementality. In the case of the inline single-objective optimization schema described in §2.3.1, we notice that all clauses that are learned by the OMT solver along the search are either spontaneously learned by the SMT solver or "artificially" introduced by the optimization engine. The former set of formulas includes either $\mathcal{T}$-lemmas (that are always valid in $\mathcal{T}$) or clauses derived from $\mathcal{T}$-lemmas and the input formula φ. The second set of formulas, instead, is comprised by clauses of the form $C_\mu \stackrel{\text{def}}{=} \mathsf{obj} < \mathsf{ub}_i$ (plus $\mathsf{obj} < \mathsf{pivot}$ and its negation in binary-search mode) that are used to drive the search towards the optimal value[14].

[14] In the case of Multiple-independent OMT, which is introduced in Section §4.6.2, C_μ is defined as $\bigvee_{\mathsf{obj}_i \in \mathcal{O}'}(\mathsf{obj}_i < \mathsf{ub}_i)$, where $\mathcal{O}'$ is the set of objectives currently being tracked.

If the underlying SMT solver is incremental, then the OMT solver can be made incremental by dropping after each optimization search all and only those clauses that were "artificially" introduced on the formula stack. The remaining clauses can instead be preserved to improve the performance of any subsequent OMT call.

Assumptions Based Incrementality. Rather than throwing away every "artificial" clause after each OMT search, an alternative solution is to leverage incremental SMT with assumptions.

With this technique, each call to the OMT solver is associated with a fresh Boolean variable A_k, that is then assumed by the underlying SMT solver at the beginning of the search. Then, whenever an "artificial" unit clause C is generated along the search, the OMT solver learns the clause $\neg A_k \vee C$ instead. For instance, $C_\mu \stackrel{\text{def}}{=} (\mathsf{obj} < \mathsf{ub}_i)$ gets replaced by $C^*_\mu \stackrel{\text{def}}{=} \neg A_k \vee (\mathsf{obj} < \mathsf{ub}_i)$. In any subsequent call to the OMT solver, the Boolean variable A_k is no longer assumed and therefore any "artificial" clause created at this stage is no more active.

Compared with the *stack-based incrementality*, this approach has a twofold advantage that boost the performance even further. First, similarly to the other technique, it allows one to reuse any $\mathcal{T}$-lemma (and derived clause) that is learned by the underlying SMT solver. Second, it also prevents any clause of the form $\neg(\mathsf{obj} < \mathsf{ub}_i) \vee C$ —which may be created after learning $C_\mu \stackrel{\text{def}}{=} (\mathsf{obj} < \mathsf{ub}_i)$— from being discarded. In any subsequent call to the OMT solver, these clauses are activated as soon as a clause of the form $(\mathsf{obj} < val)$ is learned, where val is a constant smaller or equal ub_i. This happens whenever the current minimum of obj becomes smaller or equal ub_i again, and also when the pivot value of a binary-search step is picked to be smaller or equal ub_i.

Experimental Evaluation. In Section §6.5, we show the benefits of Incremental OMT with an experiment that compares it with the non-incremental solving of a number of OMT formulas representing the same problem.

4.6 Multi-Objective Optimization

In Optimization Modulo Theories, multiple objective functions obj_i can be defined within the same input formula φ, depending on the requirements of the target application.

Definition 4.6.1. *(Multi-Objective OMT). Let φ be a ground SMT formula and $\mathcal{O}$ be a set of N objective functions $\{\mathsf{obj}_1, ..., \mathsf{obj}_N\}$, where each obj_i is defined over terms in φ. We call the pair $\langle \varphi, \mathcal{O} \rangle$ a* Multi-Objective OMT problem.

Remark 4.6.1. In general, the set of objectives $\{\mathsf{obj}_1, ..., \mathsf{obj}_N\}$ in a Multi-Objective OMT problem $\langle \varphi, \mathcal{O} \rangle$ does not need to be of homogeneous, so that each obj_i can be indifferently an $\mathcal{LRA}$, an $\mathcal{LIA}$, an $\mathcal{LIRA}$, a $\mathcal{PB}$, a MAXSMT, a $\mathcal{BV}$ or a $\mathcal{FP}$ objective.

In the following, we briefly examine how multiple objectives can be efficiently combined with one another in a MINMAX/MAXMIN problem (§4.6.1). Then, we provide an overview of the main multi-objective optimization schemes that are supported by OMT solvers. These include multiple-independent optimization (§4.6.2), lexicographic optimization (§4.6.3) and Pareto optimization (4.6.4).

Remark 4.6.2. (Objective Combination) We note that the simplest way to combine multiple objectives with one another is to use SMT-LIBV2language constructs of the appropriate type to build more complex goals. For example, given the set $\mathsf{obj}_1, ..., \mathsf{obj}_N$ of $\mathcal{LIRA}$-objectives, a recurrent requirement is that of optimizing an arbitrary weighted sum of these goals, i.e., $\sum_i w_i \cdot \mathsf{obj}_i$. We assume the reader is familiar with this approach, and do not discuss it any further.

4.6.1 MINMAX/MAXMIN Combination

Definition 4.6.2. *(*MINMAX *OMT,* MAXMIN *OMT). Given an SMT formula φ and a set of goals $\{\mathsf{obj}_1, ..., \mathsf{obj}_N\}$, we call* MINMAX *OMT the problem of finding the model $\mathcal{M}$ that minimizes the maximum value of the obj_i's.*

Dually, we call MAXMIN *OMT the problem of finding the model $\mathcal{M}$ that maximizes the minimum value of all obj_i.*

As described in [ST15b], such a MINMAX OMT problem can be encoded into a single-objective OMT instance $\langle \varphi', \mathsf{obj}' \rangle$, where

$$\varphi' \stackrel{\text{def}}{=} \varphi \wedge \bigwedge_{i=1}^{N} (\mathsf{obj}_i \leq \mathsf{obj}') \tag{4.18}$$

and obj' is a fresh variable of the same type as the goals in $\{\mathsf{obj}_1, ..., \mathsf{obj}_N\}$. (The encoding of MAXMIN OMT is dual.)

4.6.2 Multiple-Independent Optimization

We define the *Multiple-Independent OMT problem* as follows.

Definition 4.6.3. *(Multiple-Independent OMT [LAK$^+$14, BP14, BPF15, ST15b, ST15c]). Let $\langle \varphi, \mathcal{O} \rangle$ be a multi-objective OMT problem, where φ is a ground SMT formula and $\mathcal{O} \stackrel{def}{=} \{\mathsf{obj}_1, ..., \mathsf{obj}_N\}$, as in Definition 4.6.1. We call* Multiple-Independent OMT problem*, a.k.a* Boxed OMT problem *[BP14, BPF15], the problem of finding* in one single run *a set of models $\{\mathcal{M}_1, ..., \mathcal{M}_N\}$ such that each $\mathcal{M}_i$ makes obj_i minimum on the common formula φ. We use the notation $\langle \varphi, \mathcal{O} \rangle_\Box$ to formally indicate a* Multiple-Independent OMT problem*.*

Remark 4.6.3. Solving a Multiple-Independent OMT problem $\langle \varphi, \{\mathsf{obj}_1, ..., \mathsf{obj}_N\} \rangle_\Box$ is akin to independently solving N single-objective OMT problems $\langle \varphi, \mathsf{obj}_1 \rangle, ..., \langle \varphi, \mathsf{obj}_N \rangle$. However, the former allows for factorizing the search and thus obtaining a significant performance boost when compared to the latter approach [LAK$^+$14, BP14, ST15c].

In [ST15c], we extended the single-objective optimization procedures of [ST12, ST15a] to deal with a Multiple-Independent OMT problem $\langle \varphi, \mathcal{O} \rangle_\Box$ where each obj_i was a $\mathcal{LIRA}$ goal. Hereafter, we describe the same optimization procedure that was presented in [ST15c] and implemented in OPTIMATHSAT, noting that in this context we consider goals of any type, that is, each obj_i can be indifferently an $\mathcal{LRA}$, an $\mathcal{LIA}$, an $\mathcal{LIRA}$, a $\mathcal{PB}$, a MAXSMT, a $\mathcal{BV}$ or a $\mathcal{FP}$ objective[15]. A similar approach is implemented in Z3 [BP14, BPF15].

Input. The algorithm takes as input a pair $\langle \varphi, \mathcal{O} \rangle_\Box$, where $\mathcal{O} \stackrel{\text{def}}{=} \{\mathsf{obj}_1, ..., \mathsf{obj}_N\}$, and returns a list of minimum-cost models $\{\mathcal{M}_1, ..., \mathcal{M}_N\}$ and the corresponding list of minimum values $\{\mathsf{ub}_1, ..., \mathsf{ub}_N\}$.

Initialization. We assume, without any loss of generalization, that the algorithm starts after a round of the SMT solver found a satisfiable model $\mathcal{M}$ for the input formula φ. This allows us to disregard the case in which φ is unsatisfiable and, more importantly, to avoid introducing any special notation for dealing with the domain differences among $\mathcal{LIRA}$, $\mathcal{BV}$ and $\mathcal{FP}$ objectives[16]. Therefore, the algorithm is initialized with a list of watched objectives $\mathcal{O}' = \mathcal{O}$ and, for every $\mathsf{obj}_i \in \mathcal{O}$, an initial model $\mathcal{M}_i$ set to be equal $\mathcal{M}$ and an initial upper bound ub_i set to be equal $\mathcal{M}(\mathsf{obj}_i)$.

[15]Here, we assume that we do not separately apply any of the specialized optimization procedures among those described for $\mathcal{BV}$ (§4.3), $\mathcal{FP}$ (§4.4) and $\mathcal{PB}$/MAXSMT (§4.2.2) goals. Furthermore, we assume that any $\mathcal{PB}$/MAXSMT objective has been previously encoded into a $\mathcal{LIRA}$ goal as described in §4.2.1.

[16]The symbols $-\infty$ and $+\infty$, which in [ST15c] were used to indicate the absence of a lower bound and the absence of a solution respectively, could be source of confusion when dealing with $\mathcal{FP}$ objectives, since the domain of a $\mathcal{FP}$ variable admits its own $-\infty/+\infty$ values defined in the $\mathcal{FP}$ Theory.

Main Loop. Like in the case of inline single-objective optimization (see Section §2.3.1), the optimization search proceeds by enumerating satisfiable truth assignments that propositionally satisfy both the input formula φ and any learned clause.

Each time a consistent truth assignment μ is found, and for each watched objective $\mathsf{obj}_i \in \mathcal{O}'$, the OMT solver invokes the appropriate $\mathcal{T}$-minimization procedure over μ to find the model $\mathcal{M}'_i$ and the corresponding minimum value $\mathsf{ub}'_i \stackrel{\text{def}}{=} \mathcal{M}'_i(\mathsf{obj}_i)$. If $\mathsf{ub}'_i < \mathsf{ub}_i$, then both the current upper bound ub_i and the model $\mathcal{M}_i$ are updated with the values of ub'_i and $\mathcal{M}'_i$ respectively. Additionally, in the case of a $\mathcal{LIRA}$ objective obj_i, if the latter is found to be not lower-bounded over the propositional model μ by the $\mathcal{LIRA}$-minimization procedure, then it is also removed from the list of watched objectives $\mathcal{O}'$. This is not needed for $\mathcal{BV}$ and $\mathcal{FP}$ objectives, as their domain can only contain finitely many values.

Then, the OMT solver learns the clause

$$C_\mu \stackrel{\text{def}}{=} \bigvee_{\mathsf{obj}_i \in \mathcal{O}'} (\mathsf{obj}_i < \mathsf{ub}_i) \tag{4.19}$$

and proceeds with another round of the CDCL-based SMT search looking for an improving truth assignment μ'.

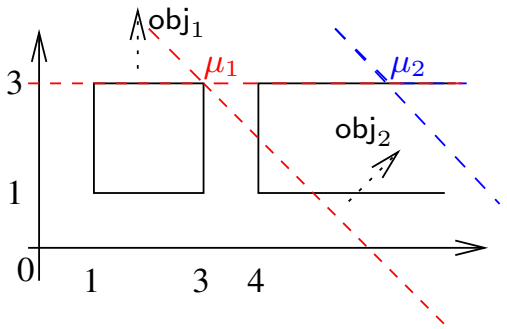
Remark 4.6.4. The clause (4.19) ensures a progress in the minimization of at least one watched objective obj_i, since by construction C_μ is such that $\mu \wedge C_\mu \models_{\mathcal{T}} \bot$. The algorithm, however, can simultaneously improve the value of *possibly many more than one* of the remaining watched objectives at each round of the OMT search.

Termination. The search terminates either when (1) $\mathcal{O}'$ is empty or when (2) the conjunction of φ with the most recent C_μ is unsatisfiable, that is, when $\varphi \wedge C_\mu \models_{\mathcal{T}} \bot$.

The argument for the termination of the multiple-independent OMT algorithm descends directly from the proof of termination in the case of single-objective optimization. In fact, the unit clauses contained in C_μ are the same as those generated in single-objective optimization. Learning this clause guarantees that, at some point of the search for an improving truth assignment μ', the SMT solver is forced to decide the literal associated to one such unit clauses, thus ensuring a progress in the minimization of the corresponding watched objective obj_i.

Example 4.6.1. *Figure 4.8 shows a toy example of multiple-independent OMT search over the pair* $\langle \varphi, \{\mathsf{obj}_1, \mathsf{obj}_2\}\rangle_\square$*, which is taken from [ST15c]. The definitions of* φ*,* obj_1 *and* obj_2 *are given in Figure 4.8.*

In a possible execution of the OMT solver, the optimization search finds the consistent truth assignment μ_1*, defined as in Figure 4.8, first. (For the sake of readability, redundant liter-*



$$
\begin{array}{rcl}
\varphi & \stackrel{\text{def}}{=} & (1 \leq y) \wedge (y \leq 3) \\
& \wedge & (((1 \leq x) \wedge (x \leq 3)) \vee (x \geq 4)) \\
& \wedge & (\mathsf{obj}_1 = -y) \wedge (\mathsf{obj}_2 = -x - y) \\
\mu_1 & \stackrel{\text{def}}{=} & \{(1 \leq y), (y \leq 3), (1 \leq x), (x \leq 3), \\
& & (\mathsf{obj}_1 = -y), (\mathsf{obj}_2 = -x - y)\} \\
\mu_2 & \stackrel{\text{def}}{=} & \{(\mathsf{obj}_1 = -y), (\mathsf{obj}_2 = -x - y), \\
& & (1 \leq y), (y \leq 3), (x \geq 4), \\
& & (\mathsf{obj}_2 < -6)\}
\end{array}
$$

Figure 4.8: An example of multiple-independent OMT search with OPTIMATHSAT on the same toy example presented in [LAK+14] (Figure taken from [ST15c]).

als were removed from each μ_i, e.g. "$\neg(x \geq 4)$" from μ_1.) The result of invoking the $\mathcal{T}$-minimization procedure over μ_1 is a pair of new upper bounds $\mathsf{ub}_1 = -3$ and $\mathsf{ub}_2 = -6$ for the watched objectives obj_1 and obj_2 respectively. Thus, the clause $(\mathsf{obj}_1 < -3) \vee (\mathsf{obj}_2 < -6)$ is learned and the OMT solver resumes the search for a satisfiable truth assignment, eventually finding μ_2. With the aid of the $\mathcal{T}$-minimization procedure invoked over μ_2, the OMT solver is then able to determine that obj_2 is lower unbounded, so that it can be removed from the current list of watched objectives $\mathcal{O}*$. The upper bound for obj_1, instead, remains unchanged. The new unit clause $(\mathsf{obj}_1 < -3)$ is learned, causing an inconsistency on the formula stack and thus terminating the optimization search. At the end of the search, the optimal values for obj_1 and obj_2 are -3 and $-\infty$ respectively.

In an alternative execution, the consistent truth assignment $\mu_2 \setminus \{(\mathsf{obj}_2 < -6)\}$ is found first, so that the solutions $\mathsf{obj}_1 = -3$ and $\mathsf{obj}_2 = -\infty$ are immediately obtained and the unit clause $(\mathsf{obj}_1 < -3)$ is learned. As a consequence, the optimization search terminates without generating μ_1. ◇

Improvements. In [ST15c], we proposed three enhancements to the above procedure.

First, at each round of optimization search, only the most recently generated clause C_μ is kept on the formula stack, while those generated previously are safely dropped from the formula stack. This is because the clause C_μ becomes stronger as the search makes some progress towards the optimal solution.

Second, before the $\mathcal{T}$-minimization procedure is invoked on μ, we use the corresponding $\mathcal{T}$-solver to check whether the constraint $(\mathsf{obj}_i < \mathsf{ub}_i)$ is satisfiable on μ. If that is not the case, then there is no chance —with the current truth assignment— of improving the value of obj_i and we avoid calling the $\mathcal{T}$-minimization procedure.

Third, when the upper bound value ub_i of a watched objective obj_i is improved to the new

value ub'_i, we also learn the $\mathcal{T}$-valid clause $(\mathsf{obj}_i < \mathsf{ub}_i) \rightarrow (\mathsf{obj}_i < \mathsf{ub}'_i)$. Then, as soon as $(\mathsf{obj}_i < \mathsf{ub}'_i)$ is assigned to true, this allows for "activating" any clause in the form $\neg(\mathsf{obj}_i < \mathsf{ub}_i) \vee C$ that was automatically learned in a previous round of the CDCL-based SMT search.

Comparison with [LAK$^+$14]. The optimization approach for multiple-independent OMT described above, adopted by both OPTIMATHSAT and Z3 [BP14, BPF15, ST15b, ST15c], is substantially different from that presented by Li et al. in [LAK$^+$14] and described in Section §2.3.1.

In our approach, each time a new truth assignment is generated by the CDCL-based SMT search, the OMT solver uses a dedicated $\mathcal{T}$-minimization procedure to eagerly improve the upper bound value of as many objectives obj_i as possible.

In comparison, the algorithm described in [LAK$^+$14], implemented on top of the Z3 SMT solver, relies on set of inference rules to either (1) force an improvement of the current solution along some objective direction or (2) prove that some objective is unbounded. Hence, in contrast with our approach, each inference rule application only affects one objective obj_i at a time.

Moreover, while OPTIMATHSAT can handle multiple-independent OMT of a mixed set of $\mathcal{LRA}$, $\mathcal{LIA}$, $\mathcal{LIRA}$, $\mathcal{BV}$, $\mathcal{FP}$, $\mathcal{PB}$ and MAXSMT objectives indifferently, the tool presented in [LAK$^+$14] deals with $\mathcal{LRA}$ objectives only.

Experimental Evaluation. In Section §6.5, we demonstrate the benefits of the Multiple-Independent optimization technique described here with an experimental evaluation that compares it with Incremental OMT, and also with the sequential solving of single-objective instances representing the same problem.

4.6.3 Lexicographic Optimization

We define the *Lexicographic OMT problem* as follows.

Definition 4.6.4. *(Lexicographic OMT [BP14, BPF15, ST15b, ST15c]). Let $\langle \varphi, \mathcal{O} \rangle$ be a multi-objective OMT problem, where φ is a ground SMT formula and $\mathcal{O} \stackrel{def}{=} \{\mathsf{obj}_1, ..., \mathsf{obj}_N\}$, as in Definition 4.6.1. We call* Lexicographic OMT problem*, the problem of finding the model $\mathcal{M}$ that satisfies φ and makes each obj_i minimum in decreasing order of priority. We use the notation $\langle \varphi, \mathcal{O} \rangle_{\mathcal{L}}$ to formally indicate a* Lexicographic OMT problem*.*

Lexicographic optimization was first introduced in OMT by Bjorner et al. in [BP14], where they illustrated the general problem and provided an implementation in an optimizing version

function UNIFIED_LEX(φ, obj_1, ..., obj_N)

1: $\langle res, \mathcal{M}' \rangle \coloneqq$ SMT.CHECK(φ)
2: **if** $(res ==$ UNSAT$)$ **then**
3: **return** $\langle$UNSAT$, \emptyset\rangle$
4: **let** $b_1, ..., b_{N+1}$ **be** fresh Boolean literals
5: **repeat**
6: $\mathcal{M} \coloneqq \mathcal{M}'$
7: $\alpha \coloneqq \{b_1, \neg b_{N+1}\} \cup$
$\bigcup_{i=1}^{i=N} \{b_i \rightarrow (\mathsf{obj}_i \leq \mathcal{M}[\mathsf{obj}_i] \wedge (\mathsf{obj}_i < \mathcal{M}[\mathsf{obj}_i] \vee b_{i+1}))\}$
8: $\langle res, \mathcal{M}' \rangle \coloneqq$ SMT.CHECK_UNDER_ASSUMPTIONS(φ, α)
9: **until** $(res ==$ UNSAT$)$
10: **return** $\langle$SAT$, \mathcal{M}\rangle$

Figure 4.9: The lexicographic OMT algorithm based on unified search implemented in OPTIMATHSAT.

of Z3. In the following, we describe in detail two approaches for lexicographic optimization with OMT solvers that have been implemented in OPTIMATHSAT.

Remark 4.6.5. As observed in remark 4.6.1, multi-objective OMT does not constrain the goals in the set $\{\mathsf{obj}_1, ..., \mathsf{obj}_N\}$ to have all the same type. Therefore, in this section we consider approaches that are applicable to heterogeneous sets of goals only and do not contemplate some other specialized techniques that are applicable when all objectives have the same type (e.g. in the case of lexicographic optimization of a set of MAXSMT goals, some of the lexicographic extensions to MAXSAT presented in [MSAGL11] could be adopted by OMT solvers).

In this section, we describe two procedures for lexicographic optimization, namely UNIFIED_LEX and ITERATED_LEX. The first algorithm can be implemented on top of any SMT solver, but it is limited by the fact that the optimization search is not guaranteed to terminate in the case of $\mathcal{LRA}$ goals —due to Zeno-ness effects [ST12, ST15a]— and also in the case of unbounded $\mathcal{LIA}$ goals. Conversely, the second algorithm has no such limitations, but it can only be implemented on top of an OMT solver with some single-objective optimization procedure for the cost function at hand. We provide a more extensive comparison among the two approaches at the end of this section.

UNIFIED_LEX. The algorithm, shown in Figure 4.9, takes as input the SMT formula φ and the sorted list of objectives $\mathsf{obj}_1, ..., \mathsf{obj}_N$.

The procedure starts by searching an initial model $\mathcal{M}$ for the input formula φ (line 1). If no such model exists, i.e. φ is unsatisfiable, then the solver returns UNSAT and terminates (lines 2-3). Otherwise, the solver instantiates a list $b_1, ..., b_{N+1}$ of $N + 1$ fresh Boolean literals (line 4). Then, it enters a loop which purpose is to improve the current model $\mathcal{M}$ up until when the optimal solution is found. The algorithm guarantees that at each iteration of the loop the value of at least one objective function obj_i is improved with respect to the previous model. Although the optimization search is not required to advance in any fixed order, it forbids any improvement of value for a goal obj_i that requires worsening the value of some other goal obj_j such that $j < i$.

At beginning of each iteration of the loop, the current model $\mathcal{M}$ is updated with the most recently found one (line 6). Then, the procedure instantiates a new set of assumptions α (line 7) with the literal b_1, the negated literal $\neg b_{N+1}$ and N constraints. Each of these constraints requires that, when the Boolean literal b_i is decided, (1) the value of the objective function $cost_i$ must be smaller or equal than its previous value in the most recent model $\mathcal{M}$ and (2) if the value of $cost_i$ cannot be improved with respect to its previous value in the most recent model $\mathcal{M}$, then the Boolean literal b_{i+1} is implied. The Boolean literal b_1 is included in the set of assumptions to ensure that the first constraint is activated, while the negation of the Boolean literal b_{N+1} serves the purpose of signaling the end of the optimization search, as it causes an inconsistency as soon as b_{N+1} becomes implied. Intuitively, this hierarchical chain of constraints in α requires an improvement on the value of objective obj_i or, alternatively, it allows for maintaining the same value for obj_i while recursively shifting the same obligations on the next objective obj_{i+1} in the list, up until the value of some obj_j is actually improved or the last Boolean literal b_{N+1} is implied, meaning that the value of every objective cannot be improved any further.

Then, algorithm checks the satisfiability of the input formula φ under the assumptions of set of constraints α together with the first literal in b, that activates the hierarchy of constraints in α (line 8). The call to the underlying SMT solver returns SAT if there exists an assignment of values to the objectives $\mathsf{obj}_1, ..., \mathsf{obj}_N$ that is lexicographically better than the one in $\mathcal{M}$. In this case, the algorithm proceeds with the next iteration of the loop, looking for a further improving solution. Otherwise, when the call to the underlying SMT solver returns UNSAT, the optimal model $\mathcal{M}$ is returned after breaking the loop (lines 9-10).

ITERATED_LEX [ST15c]. The algorithm, shown in Figure 4.10, takes as input the SMT formula φ and the sorted list of objectives $\mathsf{obj}_1, ..., \mathsf{obj}_N$.

Overall, the optimization search proceeds in rounds, starting from the goal with highest priority, namely, obj_1. At each round, the OMT solver finds the optimal model $\mathcal{M}$ for obj_i over the input formula φ conjoined with the —initially empty— list of assumptions α (line 3). Depending on the result of this check, there can be three cases.

function ITERATED_LEX(φ, obj_1, ..., obj_N)

1: $\alpha \coloneqq \emptyset$
2: **for** $i \coloneqq 1$ **up to** N **do**
3: $\quad \langle res, \mathcal{M} \rangle \coloneqq$ OMT.MINIMIZE($\varphi \cup \alpha, \mathsf{obj}_i$)
4: $\quad$ **if** $(res ==$ UNSAT$)$ **then**
5: $\quad\quad$ **return** $\langle$UNSAT$, \emptyset\rangle$
6: $\quad$ **else if** (IS_MINUS_INF($\mathcal{M}[\mathsf{obj}_i]$)) **then**
7: $\quad\quad$ **return** $\langle$SAT$, \mathcal{M}\rangle$
8: $\quad \alpha \coloneqq \alpha \cup \{\mathsf{obj}_i = \mathcal{M}[\mathsf{obj}_i]\}$
9: $\quad$ **if** $(i < N)$ **then**
10: $\quad\quad \alpha \coloneqq \alpha \cup \{\mathsf{obj}_{i+1} \leq \mathcal{M}[\mathsf{obj}_{i+1}]\}$
11: **return** $\langle$SAT$, \mathcal{M}\rangle$

Figure 4.10: The lexicographic OMT algorithm based on iterated optimization implemented in OPTIMATHSAT.

If the set of constraints in unsatisfiable, then the input formula φ is unsatisfiable and the algorithm stops with UNSAT as returned value (lines 4-5). It is worth noting that (1) since α is initially empty, the first call to the OMT solver can only return UNSAT if φ is inconsistent (2) if a round is found to be satisfiable and it returns a model $\mathcal{M}$, then the following round cannot return UNSAT because no constraint added to α removes $\mathcal{M}$ from the feasible space.

If the goal obj_i is found to be not lower-bounded, then the algorithm returns SAT along the most recently found model $\mathcal{M}$ (lines 6-7), because it is not possible to improve the solution any further.

If, instead, the optimal value of obj_i is finite, then the algorithm adds a constraint to α that binds the value of $cost_i$ to be equal to the one it has in the current model $\mathcal{M}$ (line 8). Furthermore, it extends α with a *look-ahead constraint* that forces the next objective obj_{i+1} in the list (if any) to be smaller or equal its current model value in $\mathcal{M}$ (lines 9-10). The benefit of learning this unit clause is that it restricts the feasible search space for the underlying OMT solver and, given a $\mathcal{PB}$/MAXSMT goal, it can also trigger any sorting network circuit that may be added to the input problem (see Section §4.2.1).

Algorithms comparison. The main merit of the UNIFIED_LEX procedure is that it can be implemented on top of any, preferably incremental, SMT solver. Moreover, it can be used to target any *bounded-domain* objective obj_i that belongs to a Theory $\mathcal{T}$ that is not *dense*, that is, it is not true that given any pair of value $\langle a, b\rangle$ in $\mathcal{T}$ there exists a third value c such that $a < c < b$. Both of these constraints are required to guarantee the termination of the procedure,

which could otherwise result in an infinite chain of solutions, possibly with an infinitesimal improvement at each step, and not terminate.

Conversely, the ITERATED_LEX algorithm has no such limitations, and can be executed on any input problem as long as the OMT solver can handle the optimization of obj_i in single-objective mode using, for example, the inline optimization-search schema described in Section §2.3.1. In this case, compared to the UNIFIED_LEX algorithm, the OMT solver benefits from the availability of a $\mathcal{T}$-minimization procedure as it can help reducing the number of explored solutions and it allows one to deal with both unbounded objectives and dense Theories (e.g. $\mathcal{LRA}$). In addition, if an objective obj_i is given an initial lower bound, it can benefit from using either the binary- or adaptive-search modes to further reduce the number of explored solutions. It is worth noting that the OMT solver is not limited to using the CDCL-based optimization schema, but it can also apply any specialized optimization procedure that is made available for a given objective obj_i. This includes OBV-WA/OBV-BS for $\mathcal{BV}$ goals (see Section §4.3), OFP-BS for $\mathcal{FP}$ objectives (see Section §4.4) and MAXRES for $\mathcal{PB}$/MAXSMT goals (see Section §4.2.2).

4.6.4 Pareto Optimization

Pareto optimization with OMT made its first official appearance in the paper [BP14], where Bjorner et al. described the general Pareto optimization problem and provided the pseudocode of a procedure for dealing with it based on the *Guided Improvement Algorithm* (GIA) presented in [REJ09]. This procedure was implemented in the Z3 OMT solver.

In order to define the *Pareto OMT problem*, we define the concepts of *Pareto domination* and *Pareto Optimality* first. Hereafter, without any loss of generality, we assume that every cost function is subject to minimization.

Definition 4.6.5. *(Pareto domination). Let φ be a ground SMT formula and* obj *be an objective function. Then, given a pair of models $\langle \mathcal{M}_i, \mathcal{M}_j \rangle$ such that both models satisfy φ, we say that a model $\mathcal{M}_j$* Pareto dominates *$\mathcal{M}_i$ if $\forall u. \mathcal{M}_j(\mathsf{obj}_u) \leq \mathcal{M}_i(\mathsf{obj}_u)$ and $\exists v. \mathcal{M}_j(\mathsf{obj}_v) < \mathcal{M}_i(\mathsf{obj}_v)$.*

Definition 4.6.6. *(Pareto Optimality). Given a multi-objective OMT problem $\langle \varphi, \mathcal{O} \rangle$, where φ is a ground SMT formula and $\mathcal{O} \stackrel{def}{=} \{\mathsf{obj}_1, ..., \mathsf{obj}_N\}$, is a set of N objective functions, as in Definition 4.6.1. A model $\mathcal{M}_i$ of φ is said to be* Pareto optimal *if and only if there does not exist a model $\mathcal{M}_j$ of φ such that M_j* Pareto dominates *$\mathcal{M}_k$.*

The collection of all Pareto optimal models is called Pareto front.

function GUIDED_IMPROVEMENT_ALG(φ, obj_1, ..., obj_N)

1: $\alpha \coloneqq \emptyset$
2: **while** $true$ **do**
3: INPLACE_SHUFFLE($\mathsf{obj}_1, ..., \mathsf{obj}_N$)
4: $\langle res, \mathcal{M} \rangle \coloneqq$ OMT.PARTIAL_LEX($\varphi \cup \alpha, \mathsf{obj}_1, ..., \mathsf{obj}_N$)
5: **if** ($res ==$ UNSAT) **then**
6: **return** $\langle$UNSAT, $\emptyset\rangle$
7: **while** ($res ==$ SAT) **do**
8: $c_1 \coloneqq \bigwedge_{i=1}^{N} \mathsf{obj}_i \leq \mathcal{M}[\mathsf{obj}_i]$ // no obj_i worse
9: $c_2 \coloneqq \bigvee_{i=1}^{N} \mathsf{obj}_i < \mathcal{M}[\mathsf{obj}_i]$ // some obj_i better
10: $\langle res, \mathcal{M} \rangle \coloneqq$ OMT.PARTIAL_LEX($\varphi \cup \alpha \cup \{c_1, c_2\}, \mathsf{obj}_1, ..., \mathsf{obj}_N$)
11: $\alpha \coloneqq \alpha \cup \{c_2\}$
12: **yield** $\langle$SAT, $\mathcal{M}\rangle$

Figure 4.11: The callback version of the *Guided Improvement Algorithm* for Pareto OMT implemented in OPTIMATHSAT and based on [REJ09, BP14].

Definition 4.6.7. *(Pareto OMT [BP14, BPF15]). Let $\langle \varphi, \mathcal{O} \rangle$ be a multi-objective OMT problem, where φ is a ground SMT formula and $\mathcal{O} \stackrel{def}{=} \{\mathsf{obj}_1, ..., \mathsf{obj}_N\}$, as in Definition 4.6.1. We call* Pareto OMT problem*, the problem of finding a (possibly infinite) set of models $\{\mathcal{M}_1, ..., \mathcal{M}_M\}$ of φ that belong to the Pareto front. We use the notation $\langle \varphi, \mathcal{O} \rangle_{\mathcal{P}}$ to formally indicate a* Pareto OMT problem*.*

In the following, we describe two procedures for dealing with Pareto OMT problems. The first algorithm is an implementation of the *Guided Improvement Algorithm* (GIA) that is made available in OPTIMATHSAT and it is similar to the one implemented in Z3 [BP14]. The second procedure is a novel approach, also based on the general GIA schema, that leverages the availability of a lexicographic optimization procedure to overcome some limitations of the first solution.

Guided Improvement Algorithm [REJ09, BP14].

In the following, we describe the implementation of the *Guided Improvement Algorithm* [REJ09] in OPTIMATHSAT. The algorithm follows in the footsteps of Z3's approach as it was presented by Bjorner et al. in [BP14], with only minor differences.

The algorithm, shown in Figure 4.11, takes as input an SMT formula φ plus a set of objectives $\{\mathsf{obj}_1, ..., \mathsf{obj}_N\}$, and it starts with an empty list of assumptions α.

Each iteration of the main loop from line 2 to line 12 can either yield a Pareto-optimal model $\mathcal{M}$ (line 12) or return UNSAT (lines 5-6), which can either mean that the input formula φ is unsatisfiable as a whole (at the first iteration), or that the whole Pareto front has been explored (in any subsequent round).

Remark 4.6.6. We note that, in OPTIMATHSAT, unlike Z3, we first shuffle the set of objectives $\{\mathsf{obj}_1, ..., \mathsf{obj}_N\}$ at the beginning of each round (line 3). On the one hand, this encourages the OMT solver to explore the Pareto front from different directions rather than choosing subsequent solutions from the same neighborhood, which can be useful when the Pareto front contains infinitely-many elements. On the other hand, exploring from different directions results in a fragmented feasible search-space that causes additional overhead in the long run. Still, we consider this performance trade-off acceptable.

OPTIMATHSAT invokes PARTIAL_LEX() over the input formula φ extended with the set of assumptions α to find an initial model $\mathcal{M}$ that can later on be improved according to the *Guided Improvement Algorithm* strategy. The procedure PARTIAL_LEX() is such that given an input formula φ, it looks for an arbitrary (and complete) propositional truth assignment μ that satisfies φ first, and then computes the lexicographic-optimal model $\mathcal{M}$ corresponding to $\varphi \cup \mu$ using only the minimization procedure inside each $\mathcal{T}$-Solver. If every input objective obj_i has a bounded optimal solution, this has two benefits. First, it reduces the number of iterations of the inner loop at lines 7-10, since it forces the OMT solver to generate a new truth assignment μ' at each step. Second, it avoids *Zeno-ness* behavior when some objective obj_i belongs to the Theory of $\mathcal{LRA}$ or mixed $\mathcal{LIRA}$. If the result of PARTIAL_LEX() is UNSAT, the procedure terminates immediately (lines 5-6).

Otherwise, the OMT solver is given the task of iteratively improving the initial model $\mathcal{M}$ to obtain a Pareto-optimal solution (lines 7-10). To do so, it creates two constraints c_1 and c_2 which require that the value of every obj_i is kept at least as "good" as in the current model $\mathcal{M}$ (c_1), and also that the value of some obj_i is "improved" along its optimization direction (c_2). In the subsequent call to PARTIAL_LEX() (line 10), the combination of c_1 and c_2, forces the OMT solver to search for a model $\mathcal{M}'$ that Pareto-dominates $\mathcal{M}$.

The inner loop terminates when the OMT solver finds a model $\mathcal{M}$ such that there exists no other $\mathcal{M}'$ that dominates it. In this case, the Pareto-optimal model $\mathcal{M}$ is added to the set of Pareto-front solutions (line 12). Before looking for another Pareto-optimal model $\mathcal{M}'$, the set of assumptions α is extended with the most recently generated constraint c_2 to remove $\mathcal{M}$ from the feasible space.

Termination. The outer loop in the *Guided Improvement Algorithm* depicted in Figure 4.11 is not guaranteed to terminate when there are infinitely-many Pareto-optimal models $\mathcal{M}$ to enumerate. This can be the case when obj_i is either an unbounded $\mathcal{LIA}$ objective or a (possibly bounded) mixed Rational/Integer goal.

Remark 4.6.7. Moreover, the inner loop at lines 7-10 is also not guaranteed to terminate when it is always possible to find a model $\mathcal{M}'$ such that $\mathcal{M}' \models \varphi$ and $\mathcal{M}'$ Pareto-dominates the current model $\mathcal{M}$.

We illustrate this scenario with the following toy example.

Example 4.6.2. *Consider the case of a Pareto OMT problem* $\langle \mathsf{obj}_1 = \mathsf{obj}_2, \mathsf{obj}_1, \mathsf{obj}_2 \rangle_{\mathcal{P}}$. *The execution of* PARTIAL_LEX *at line* 4 *of the algorithm depicted in Figure 4.11 returns a model* $\mathcal{M}$ *in which both* obj_1 *and* obj_2 *are assigned an arbitrary large, but still* finite*, representative value* ub. *Assume, for example, that* $\mathsf{ub} \stackrel{def}{=} -10^9$*, then the solver instantiates* $c_1 \stackrel{def}{=} \mathsf{obj}_1 \leq -10^9 \wedge \mathsf{obj}_2 \leq -10^9$ *and* $c_2 \stackrel{def}{=} \mathsf{obj}_1 < -10^9 \vee \mathsf{obj}_2 < -10^9$. *Neither of these constraints prevents the OMT solver from generating, at the next iteration, a new model* $\mathcal{M}'$ *in which the model value of both* obj_1 *and* obj_2 *has been decreased even by a fractional amount.* $\diamond$

Lexicographic Guided Improvement Algorithm.

We present here an improvement of the previous algorithm, namely *Lexicographic Guided Improvement Algorithm*, that does not suffer from non-termination in the case it is always possible to find a model $\mathcal{M}'$ such that $\mathcal{M}' \models \varphi$ and $\mathcal{M}'$ Pareto-dominates the most recently found model $\mathcal{M}$. To achieve this, it incorporates an automated mechanism for detecting unbounded objective functions and it exploits the available lexicographic optimization engine more effectively.

Unboundedness types. Given a Pareto OMT problem $\langle \varphi, \mathsf{obj}_1, ..., \mathsf{obj}_N \rangle_{\mathcal{P}}$ such that φ is a satisfiable ground SMT formula, each obj_i can be either *bounded* or *unbounded*.

An objective obj_i is said to be *bounded* if its optimal value is always finite over the input formula φ. An objective obj_i is said to be *unbounded* if there exists some truth assignment μ such that μ makes φ satisfiable and obj_i is unbounded in correspondence with μ.

Given a model $\mathcal{M}$ of φ, we say that an *unbounded* goal obj_i is *semi-bounded* if it has a finite optimal value over the single-objective OMT instance $\langle \varphi \wedge c_1, \mathsf{obj}_i \rangle$, where c_1 is defined as follows:

$$c_1 \stackrel{\text{def}}{=} \bigwedge_{j=1}^{N} \mathsf{obj}_j \leq \mathcal{M}[\mathsf{obj}_j] \tag{4.20}$$

Constraint c_1 encodes one of the two necessary and sufficient properties that must be met by a model $\mathcal{M}'$ of φ so that $\mathcal{M}'$ can be said to Pareto-dominate $\mathcal{M}$. Intuitively, an *unbounded* objective obj_i is bounded by c_1 when its value can be arbitrarily decreased only as long as the value of some other objective obj_j can be correspondingly increased.

Example 4.6.3. *Consider the case of a Pareto OMT problem* $\langle \varphi, \mathsf{obj}_1, ..., \mathsf{obj}_4 \rangle_{\mathcal{P}}$ *where* $\varphi \stackrel{def}{=} \{0 \leq \mathsf{obj}_1, (\mathsf{obj}_1 \leq 3) \rightarrow (\mathsf{obj}_2 = -\mathsf{obj}_3)\}$.

Then obj_1 *is* bounded *by the value* 0*, while* obj_2*,* obj_3 *and* obj_4 *are* unbounded*. Let* $\mu \stackrel{def}{=} \{0 \leq \mathsf{obj}_1, \mathsf{obj}_1 \leq 3, \mathsf{obj}_2 = -\mathsf{obj}_3\}$ *be a truth assignment satisfying the input formula* φ*, and let* $\mathcal{M} \stackrel{def}{=} \{\mathsf{obj}_1 = 3, \mathsf{obj}_2 = -5, \mathsf{obj}_3 = 5, \mathsf{obj}_4 = 0\}$ *be a model of* φ *corresponding to the truth assignment* μ*. Then both* obj_2 *and* obj_3 *are* semi-bounded *in correspondence with the model* $\mathcal{M}$*, because the constraint* $\mathsf{obj}_2 = -\mathsf{obj}_3$ *transforms an upper bound on either objective in a lower bound for the other goal. Neither* obj_2 *nor* obj_3 *are necessarily* semi-bounded *for a completely different choice of values. For example, consider the case of a truth assignment* $\mu \stackrel{def}{=} \{0 \leq \mathsf{obj}_1, \neg(\mathsf{obj}_1 \leq 3), \neg(\mathsf{obj}_2 = -\mathsf{obj}_3)\}$ *and its corresponding model* $\mathcal{M} \stackrel{def}{=} \{\mathsf{obj}_1 = 4, \mathsf{obj}_2 = -1, \mathsf{obj}_3 = -1, \mathsf{obj}_4 = 0\}$. $\diamond$

In the following, we illustrate the main ideas of the *Lexicographic Guided Improvement Algorithm* first, and then describe its pseudocode in detail.

Lexicographic Optimization. Let $\langle \varphi, \mathsf{obj}_1, ..., \mathsf{obj}_N \rangle_{\mathcal{P}}$ be a Pareto OMT problem such that φ is a satisfiable ground SMT formula and every obj_i is *bounded*. Then, the model found by solving the Lexicographic OMT problem $\langle \varphi, \mathsf{obj}_1, ..., \mathsf{obj}_N \rangle_{\mathcal{L}}$ is *by definition* a Pareto-optimal solution for the instance $\langle \varphi, \mathsf{obj}_1, ..., \mathsf{obj}_N \rangle_{\mathcal{P}}$.

If the size of the Pareto front is also finite, then it is possible to extract all Pareto-optimal solutions by iteratively solving a sequence of Lexicographic OMT problems $\langle \varphi \cup \alpha, \mathsf{obj}_1, ..., \mathsf{obj}_N \rangle_{\mathcal{L}}$, where the initially empty set of constraints α is extended at each iteration to block any point that is Pareto-dominated by any Pareto-optimal model $\mathcal{M}$ found so far.

If some obj_i is *unbounded*, then solving the associated Lexicographic OMT problem is not guaranteed to result in a Pareto-optimal model. In fact, as described in §4.6.3, the lexicographic optimization search yields as soon as it encounters a goal obj_i with an unbounded value.

We proceed as follows when dealing with a Pareto OMT problem that contains some *unbounded* goal obj_i. First, we sort the objectives $\mathsf{obj}_1, ..., \mathsf{obj}_N$ so that the initial k goals in the list are all *bounded* and the last $N - k$ objectives are *unbounded*. Second, we use the lexicographic optimization procedure over the sorted list of objectives to get a model $\mathcal{M}$ of φ. By construction, the returned model $\mathcal{M}$ is Pareto-optimal with respect to the initial problem restricted to the *bounded* objectives only, i.e. $\langle \varphi, \mathsf{obj}_1, ..., \mathsf{obj}_k \rangle_{\mathcal{P}}$. We use the model $\mathcal{M}$ and the Equation (4.20)

function LEX_GUIDED_IMPROVEMENT_ALG(φ, obj_1, ..., obj_N)

1: $\alpha \coloneqq \emptyset$
2: $\beta, \gamma, res \coloneqq$ CLASSIFY_OBJS$(\varphi, \mathsf{obj}_1, ...\mathsf{obj}_N)$
3: **if** $(res ==$ UNSAT$)$ **then**
4: **return** $\langle$UNSAT$, \emptyset\rangle$
5: **while** $true$ **do**
6: SHUFFLE(β)
7: $\langle res, \mathcal{M}, unb\rangle \coloneqq$ OMT.ITERATED_LEX$(\varphi \cup \alpha, \beta, \gamma)$
8: **if** $(res ==$ UNSAT$)$ **then**
9: **return** $\langle$UNSAT$, \emptyset\rangle$
10: **if** (unb) **then**
11: $c_1 \coloneqq \bigwedge_{i=1}^{N} \mathsf{obj}_i \leq \mathcal{M}[\mathsf{obj}_i]$ // no obj_i worse
12: $\langle res, \mathcal{M}, unb\rangle \coloneqq$ OMT.ITERATED_LEX$(\varphi \cup \alpha \cup c_1, \beta, \gamma)$
13: **if** (unb) **then**
14: **return** $\langle$UNKNOWN$, \emptyset\rangle$
15: $c_2 \coloneqq \bigvee_{i=1}^{N} \mathsf{obj}_i < \mathcal{M}[\mathsf{obj}_i]$ // some obj_i better
16: $\alpha \coloneqq \alpha \cup \{c2\}$
17: **yield** $\langle$SAT$, \mathcal{M}\rangle$

Figure 4.12: The callback version of the *Lexicographic Guided Improvement Algorithm* for Pareto OMT implemented in OPTIMATHSAT.

to create a bounding box that restricts the search space to exclude any potential solution that does not Pareto-dominate with respect to model $\mathcal{M}$. As a result, this forces any *semi-bounded* —within the region delimited by the constraint— goal obj_j to immediately become bounded. The lexicographic optimization engine is then invoked once again by the OMT solver to get a new model $\mathcal{M}'$ of φ. If no objective function is still unbounded, then $\mathcal{M}'$ is Pareto-optimal for the original problem. Otherwise, $\mathcal{M}'$ is still dominated by some other model $\mathcal{M}''$ of φ. At this point, one option is to proceed with the enumeration of all the candidate solutions that Pareto-dominate $\mathcal{M}'$, that can be infinitely many. This is the approach taken by the *Guided Improvement Algorithm* implementation that was previously presented. Another possibility is to simply give up, and terminate the search with an UNKNOWN result rather than to remain stuck in an infinite loop. This gives the end-user a chance to reformulate the problem and to remove the cause of non-termination.

Algorithm. The pseudocode of the *Lexicographic Guided Improvement Algorithm* is depicted in Figure 4.12.

As usual, the procedure takes as input an SMT formula φ plus a set of objectives $\{\mathsf{obj}_1, ..., \mathsf{obj}_N\}$, and it starts with an empty list of assumptions α.

At the beginning of the search, the algorithm invokes a function CLASSIFY_OBJS() that splits the set of objectives $\{\mathsf{obj}_1, ..., \mathsf{obj}_N\}$ in two disjoint subsets β and γ (line 2). The procedure uses the engine for Multiple-independent OMT to place every *bounded* goal obj_i in the set β and the remaining ones in γ. In the case φ is detected to be unsatisfiable by CLASSIFY_OBJS(), then the algorithm terminates (lines 3-4).

If φ is satisfiable, the OMT solver enters the main loop of the algorithm (lines 5-18) that extracts, at each iteration, a unique Pareto-optimal model $\mathcal{M}$ of φ.

At the beginning of each iteration, the subset of objectives β is shuffled (line 6), for the same reason outlined in remark 4.6.6 for the *Guided Improvement Algorithm*. Then, OPTIMATHSAT invokes ITERATED_LEX() on the input formula φ extended with the set of assumptions α to find an initial model $\mathcal{M}$ of the input formula (if any). If UNSAT is returned, the procedure terminates since the complete Pareto front has already been explored (lines 8-9). Otherwise, ITERATED_LEX() found a model $\mathcal{M}$ for $\varphi \cup \alpha$. The model $\mathcal{M}$ is Pareto-optimal if the lexicographic optimization search was able to terminate without encountering any unbounded objective. In this case, $\mathcal{M}$ is added to the set of Pareto-front solutions and α is extended to remove from the feasible search-space any model $\mathcal{M}'$ that is Pareto-dominated by $\mathcal{M}$ (lines 15-17). If instead ITERATED_LEX() encountered some unbounded objective, then each obj_i is upper-bounded using its value in the model $\mathcal{M}$, so that any *semi-bounded* goal is subsequently forced to become bounded when the lexicographic engine is restarted (lines 11-12). If the formula still contains some unbounded goal obj_i, then the algorithm simply gives up and returns UNKNOWN (lines 13-14). Otherwise, it jumps at line 15, where the Pareto-optimal model $\mathcal{M}$ can then be handled as previously described.

Termination. Similarly to the original algorithm, this variant of the *Guided Improvement Algorithm* does still not terminate in the case in which there are infinitely-many Pareto-optimal models $\mathcal{M}$ to be enumerated. However, in contrast with the previous approach, each iteration of the main loop in the new algorithm is guaranteed to take only a finite amount of time. This result is a trivial consequence of having removed any (possibly) infinite loop from this part of the procedure implementation.

4.7 All-OMT

We define ALL-OMT as follows.

function ALL-OMT($\mathcal{B}$, φ, obj_1, ..., obj_N)

1: $\langle res, \mathcal{M} \rangle \coloneqq$ OMT.ITERATED_LEX($\varphi, \mathsf{obj}_1, ..., \mathsf{obj}_N$)
2: **if** $(res ==$ UNSAT$)$ **then**
3: **return** $\langle$UNSAT$, \emptyset\rangle$
4: $\alpha \coloneqq \bigcup_i (\mathsf{obj}_i = \mathcal{M}[\mathsf{obj}_i])$
5: **while** $(res ==$ SAT$)$ **do**
6: **yield** $\langle$SAT$, \mathcal{M}\rangle$
7: $\alpha \coloneqq \alpha \cup \bigvee_i (\mathcal{B}_i \neq \mathcal{M}[\mathcal{B}_i])$
8: $\langle res, \mathcal{M} \rangle \coloneqq$ SMT.CHECK_UNDER_ASSUMPTIONS(φ, α)

Figure 4.13: The basic ALL-OMT algorithm implemented in OPTIMATHSAT.

Definition 4.7.1. *(*ALL-OMT*). Let $\langle \varphi, \mathcal{O} \rangle_{\mathcal{L}}$ be a lexicographic OMT problem such that φ is a* satisfiable *ground SMT formula, $\mathcal{O} \stackrel{def}{=} \{\mathsf{obj}_1, ..., \mathsf{obj}_N\}$ and $\mathcal{M}$ is a lexicographic-optimum model that satisfies φ and makes each obj_i minimum in decreasing order of priority as in Definition 4.6.4. Given a list of Boolean predicates $\mathcal{B}$, we call* ALL-OMT *the problem of enumerating all possible assignments of values to $\mathcal{B}$ that satisfy both the input formula φ and $\bigwedge_i \mathsf{obj}_i = \mathcal{M}[\mathsf{obj}_i]$. The latter constraint ensures that the optimum value of each obj_i is preserved while enumerating valid truth assignments.*

Naturally, in the simplest case, $\mathcal{O}$ can be comprised by a single objective obj_1 and $\mathcal{M}$ is the solution of the single-objective optimization of obj_1 over φ.

Notice that, according to the previous definition, ALL-OMT is not the same as enumerating all possible models $\mathcal{M}'$ of φ that preserve the optimal solution. In fact, this case would also allow for enumerating all possible model values of any non-Boolean predicate like, for example, a $\mathcal{LIA}$ or $\mathcal{LRA}$ variable. As a matter of fact, restricting the focus on Boolean predicates allows us to guarantee the termination of the proposed ALL-OMT procedure.

Algorithm. The ALL-OMT procedure, shown in Figure 4.13, takes as input the set of interesting Boolean predicates $\mathcal{B}$, an SMT formula φ and a sorted list of objectives $\mathsf{obj}_1, ..., \mathsf{obj}_N$.

The algorithm starts by looking for a model $\mathcal{M}$ that satisfies the input formula and is the lexicographic-optimum realization of the input list of objectives $\mathsf{obj}_1, ..., \mathsf{obj}_N$ (line 1). If no such model $\mathcal{M}$ exists, then ALL-OMT terminates with UNSAT (lines 2-3). Otherwise, a set of assumptions α is initialized with a conjunction of constraints that force the value of each objective obj_i to remain equal to its current optimal value in the lexicographic-optimum model $\mathcal{M}$ (line 4).

Then, the OMT solver enters the main loop at the lines 5-8, from which it can escape only

when it has enumerated all possible solutions of the ALL-OMT problem. As a first step, the OMT solver yields the current optimum model $\mathcal{M}$ —from which an assignment of values to the set of predicates $\mathcal{B}$ can be derived— to the caller (line 6). Then, it extends the set of assumptions α with a constraint of the form $\bigvee_i(\mathcal{B}_i \neq \mathcal{M}[\mathcal{B}_i])$, that removes the current model $\mathcal{M}$ (and possibly many others) from the set of feasible solutions (line 7). At this point, the OMT solver incrementally checks the satisfiability of φ conjoined with the set of assumptions α (line 8), and then jumps at the evaluation of the main loop condition.

Termination. Given a set of predicates $\mathcal{B}$ of size k, it is trivial to see that the ALL-OMT procedure in Figure 4.13 terminates. In fact, at each iteration the set of assumptions α is extended with a constraint of the form $\bigvee_i(\mathcal{B}_i \neq \mathcal{M}[\mathcal{B}_i])$, which is guaranteed to conflict with at least one assignment of values to the set of predicates $\mathcal{B}$ that was previously satisfiable. As a result, the OMT solver can enumerate at most 2^k models of φ before the satisfiability check at line 8 is forced to return UNSAT and the whole procedure terminates.

Chapter 5

OptiMathSAT

This chapter describes the architecture and the input interfaces of OPTIMATHSAT [opt], an Optimization Modulo Theories solver based on MATHSAT5 [CGSS13b]. OPTIMATHSAT was first presented by Sebastiani and Tomasi in [ST12, ST15a], and subsequently extended by Sebastiani and Trentin in [ST15b, ST15c, ST17, ST18, TS19].

This chapter is organized as follows:

§5.1 A high-level overview of the *architecture* of OPTIMATHSAT.

§5.2 A discussion on the *compositional approach* that is adopted by OPTIMATHSAT.

§5.3 An overview of the *input/output interfaces* of OPTIMATHSAT. Sections §5.3.1 and §5.3.2 cover the two file-based interfaces of OPTIMATHSAT, the first describing the *extended* SMT-LIBV2 *syntax* and the second the MINIZINC interface of OPTIMATHSAT. Section §5.3.3, instead, covers the public *API* accessible through its library.

§5.4 Documents the *Configurable Options* of OPTIMATHSAT.

Full Disclosure. Most of the material presented here is based on the content of [ST18], that describes OPTIMATHSAT version 1.4.2. Since then, eight new updates have been released and OPTIMATHSAT reached version 1.6.2. Therefore, in this chapter we updated the content of [ST18] to reflect the latest changes in the tool.

5.1 Architecture

OPTIMATHSAT is based on MATHSAT5, and it is developed in C++. The tool is made available at the web-page `http://optimathsat.disi.unitn.it/`. The website includes

links for downloading the latest version of the tool's binary and library, compiled for Linux, Windows and Mac OS X. Moreover, it includes extensive documentation, code examples, a list of publications describing or using the tool, the preferred contact address to get in touch with the development team and more. OPTIMATHSAT inherits the license conditions of MATHSAT5, and it is thus made freely available for research and evaluation purposes only.

Figure 5.1 depicts a simplified schema of OPTIMATHSAT architecture. At a high-level, the tool takes as input a pair $\langle \varphi, \mathcal{O} \rangle$, where φ is a (ground) SMT formula and $\mathcal{O} \stackrel{\text{def}}{=} \mathsf{obj}_1, ..., \mathsf{obj}_N$ is a set of objectives. When φ is unsatisfiable, the solver returns UNSAT. Otherwise, if φ is satisfiable, OPTIMATHSAT returns SAT plus a set of models $\mathcal{M} \stackrel{\text{def}}{=} \mathcal{M}_1, ..., \mathcal{M}_N$ such that $\mathcal{M}_i$ makes obj_i optimal over φ.

OPTIMATHSAT extends the SMT Interface of MATHSAT5 with an OMT Interface that implements the optimization functionalities described in Chapter §4 while preserving the access to all functionalities of the underlying SMT solver. In other words, OPTIMATHSAT behaves as a wrapper of MATHSAT5 when none of its optimization, MAXSMT and Pseudo-Boolean extensions are used.

The CDCL/SAT engine at the core of MATHSAT5 for dealing with *Satisfiability Modulo Theories* is integrated with a flexible optimization procedure, identified with the Boxed Optimizer block in Figure 5.1, that can handle *Linear Arithmetic* (see Sections §2.3.1 and §4.1), *Bit-Vector* (see Section §4.3.1) and *Floating-Point* (see Section §4.4.1) objectives indifferently. Throughout this dissertation we often use the wording "OMT-based approach" to refer to the optimization search performed by this block, because it is based on the inline optimization schema for OMT presented in [ST12, ST15a]. When dealing with single-objective problems, the optimization search can run either in *linear-*, *binary-* or *adaptive-search* mode, as described in Section §2.3.1. When dealing with multiple objectives, in the so-called Multiple-Independent (a.k.a. Boxed) combination, the optimization search runs exclusively in *linear-search* mode according to the approach described in Section §4.6.2.

The *$\mathcal{T}$-solver* for linear arithmetic of MATHSAT5 is enhanced with the Simplex-based optimization procedure for *Linear Rational Arithmetic* described in Section §2.3.1 and with the Branch&Bound optimization procedure for *Linear Integer Rational Arithmetic* described in Section §4.1. Both procedures are visually represented with the optimizer block enclosed by the $\mathcal{LIRA}$ block in Figure 5.1. Currently, the various types of $\mathcal{BV}$-Solvers and $\mathcal{FP}$-Solvers that are made available by MATHSAT5 contain only the *stub* of an ad-hoc optimization procedure, that returns the current value of the objective function. Since at the time being these *stubs* are reserved for future use and provide no other interesting functionality, they were hidden from the general schema depicted in Figure 5.1.

The Objective block wraps the objective function to decouple the OMT-based optimization

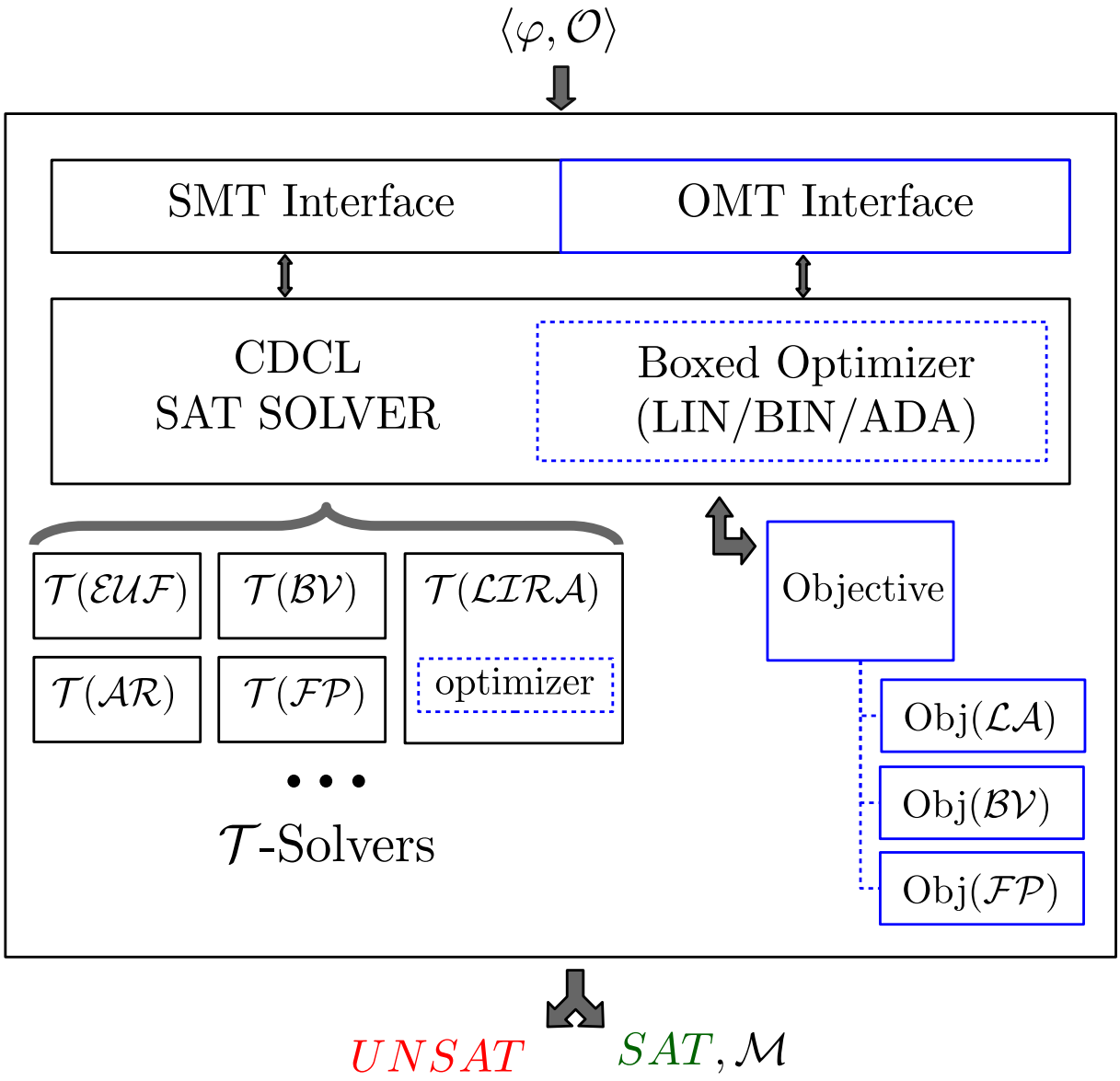


Figure 5.1: High-level overview of OPTIMATHSAT architecture.
Legend: black components are part of the underlying SMT solver, and blue ones are exclusively part of OPTIMATHSAT

search from the specific type of the objective being optimized. This modular approach makes it easy to extend the OMT-based search of OPTIMATHSAT to support new objectives types, provided that the underlying SMT solver includes a $\mathcal{T}$*-solver* for that theory.

Figure 5.2 shows a detailed view of the OMT Interface block depicted in Figure 5.1. This block contains most of the remaining functionalities introduced in OPTIMATHSAT, and it is organized as a layered stack of independent blocks.

The top-most layer contains the public input/output interface of OPTIMATHSAT, that includes the novel optimization extensions to both the External C API (see Section §5.3.3) and the SMT-LIBv2 parser (see Section §5.3.1), plus the new FLATZINC parser (see Section §5.3.2).

Underneath the top-most layer is the Optimization Context block, that integrates the novel functionalities of OPTIMATHSAT with those of MATHSAT5. In particular, it manages the stack of objective functions, the configurable options for the optimization search (see Section §5.4), the ALL-OMT algorithm described in Section §4.7, the functionality for retrieving optimum models and other implementation details related to the OMT state that are hidden to the end-user.

The Optimization Context block leverages the Sorting Network block to handle Pseudo-Boolean/MAXSMT objectives and constraints defined with the `assert-soft` command, and extends the input formula with one of the sorting network encodings described in Section §4.2.1.

The Multi-Objective Handling block is responsible for handling multi-objective OMT instances according to the multi-objective combination approach selected in the configuration[17]. The Pareto Engine block contains the *Guided Improvement Algorithm* and the *Lexicographic Guided Improvement Algorithm* described in Section §4.6.4. This block uses the functionality provided by the Lexicographic Engine, that contains both the *Unified* and *Iterated* algorithms for lexicographic optimization described in Section §4.6.3.

Notice that, when dealing with Multiple-Independent (a.k.a. Boxed) OMT or single-objective OMT instances, the objectives are handed directly to the underlying Optimization Engines block for optimization. Whenever possible, depending on the type of the objective function and on the configuration of the tool, OPTIMATHSAT invokes a specialized single-objective optimization routine over each objective individually. In the case of $\mathcal{BV}$ objectives, OPTIMATHSAT can use either the OBV-WA algorithm described in Section §4.3.2, or the OBV-BS algorithm described in Section §4.3.3. For $\mathcal{FP}$ objectives, OPTIMATHSAT can use the OFP-BS algorithm described in Section §4.4.2. In the case of compatible $\mathcal{PB}$/MAXSMT objectives, OPTIMATHSAT can use either the MAXRES engine described in Section §4.2.2, or a porting into

[17]Notice that the Linear (see Chapter §4.6), MINMAX and MAXMIN (see Section §4.6.1) combinations of objectives are directly handled by the Optimization Contex block through simple rewriting.

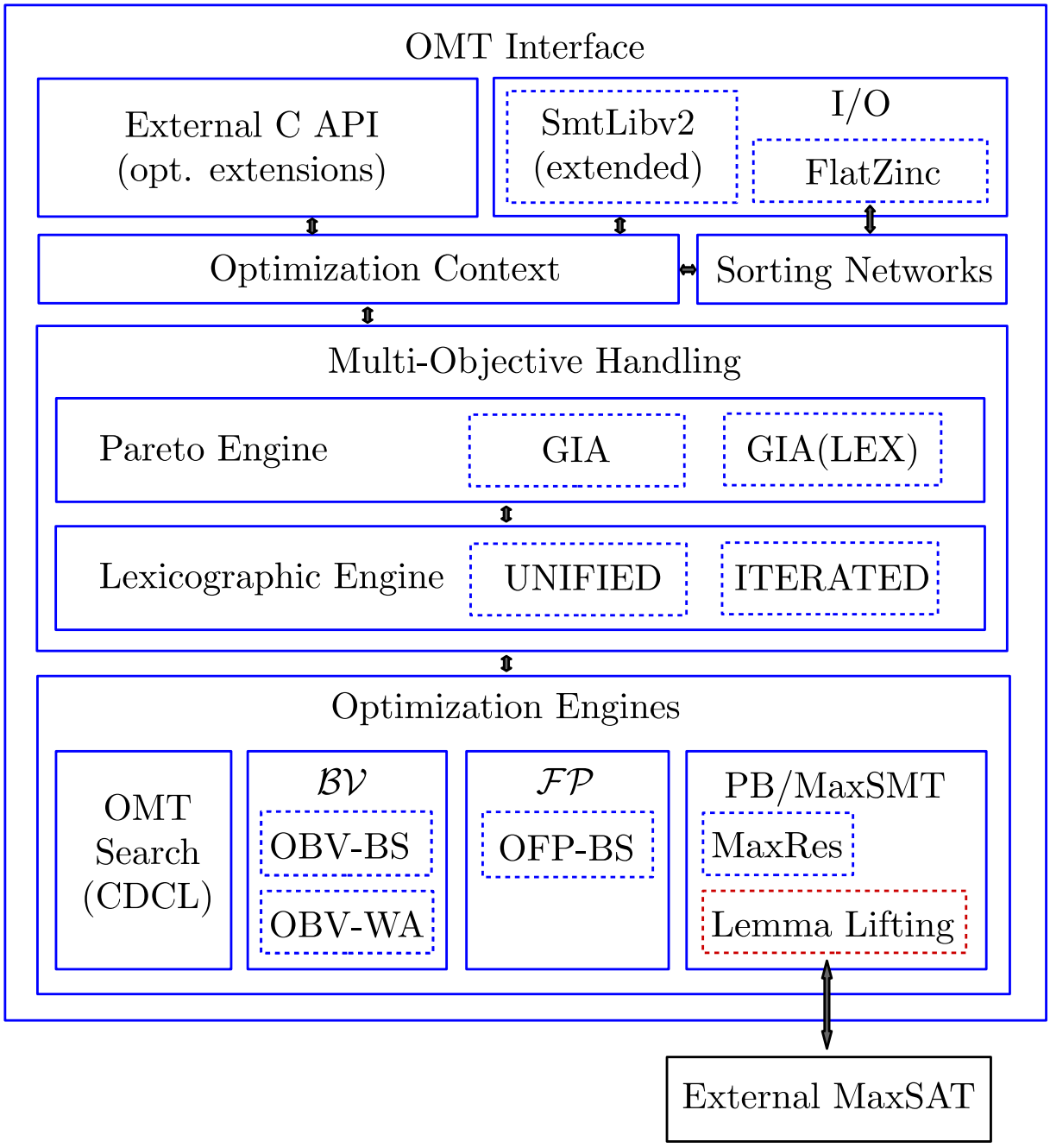


Figure 5.2: Detailed high-level overview of OPTIMATHSAT OMT Interface.
Legend: blue components are exclusively part of OPTIMATHSAT, and the only red component is an independent extension to MATHSAT5 that we ported in OPTIMATHSAT's code-base [CGSS13a]

OPTIMATHSAT of the lemma-lifting approach presented in [CGSS13a], that was originally implemented as an independent MATHSAT5 extension and requires an external MAXSAT engine. All of the remaining objectives, for which no specialized optimization routine is used, are lumped together and handed over to the *Boxed Optimizer* block embedded in the underlying CDCL sat solver shown in Figure 5.1.

5.2 Compositional Approach

As described in Section §4.2, OPTIMATHSAT provides specialized techniques for dealing with OMT($\mathcal{PB} \cup \mathcal{T}$) and MAXSMT problems. One limitation of the OMT solver is that, at the time being, it is only able to exploit these advanced techniques if Pseudo-Boolean objectives and constraints are encoded as MAXSMT, that is, in terms of soft clauses.

Some OMT applications require the ability to define objective functions in terms of complex combinations of $\mathcal{LIRA}$, $\mathcal{BV}$, $\mathcal{FP}$, $\mathcal{PB}$ and MAXSMT goals. This is the case, e.g., of *Structured Learning Modulo Theories* [TSP17] (see Section §7.2), where the objective function shown in (6.1) has both a $\mathcal{PB}$ and a $\mathcal{LIRA}$ component (see Section §6.2), or of *Linear Generalized Disjunctive Programming* (LGDP) [RG94], in which the goal combines mixed Boolean/numeric objectives.

OPTIMATHSAT deals with this necessity by adopting an *explicit and compositional definition of objectives*. This has three consequences. First, OPTIMATHSAT does not implicitly create any objective function without an explicit request by the end-user. This is in contrast with what is currently done in Z3, that defines MAXSMT goals implicitly. Second, objective functions are no longer special, independent entities separated from the rest of the formula. Instead, they are reusable objects that are treated in the same way as terms and constraints appearing in the original formula. In particular, the stack of objectives is managed analogously as the stack of formulas, meaning that issuing a `push` or a `pop` command contemporarily affects the state of both stacks. Third, OPTIMATHSAT allows for the arbitrary composition of objectives as described in Section §4.6, regardless of the fact that they are $\mathcal{LIRA}$, $\mathcal{BV}$, $\mathcal{FP}$, $\mathcal{PB}$ or MAXSMT goals. This includes both linear and MAXMIN/MINMAX compositions (§4.6.1), as well as the Multiple-Independent (§4.6.2), the Lexicographic (§4.6.3) and the Pareto (§4.6.4) optimization.

In OPTIMATHSAT, an objective can be viewed as a *term* occurring in the formula. Each objective is associated with a fresh term, of the same type, that acts as an *alias* for the objective itself. OPTIMATHSAT guarantees that the *alias* evaluates to the same value of the objective function for the whole duration of the optimization search, up until when the goal is popped from the objectives stack.

Clearly, imposing constraints on objective functions affects the space of feasible solutions

and can therefore have an impact on the outcome of the optimization search (i.e. the optimal solution). Therefore, OPTIMATHSAT protects the end-user from an inadvertent improper use of an objective *alias* by masking it with an internal variable. Whenever necessary, this precaution can be circumvented by explicitly assigning a label to the objective function in the extended SMT-LIBV2 language, e.g.

```
(minimize (+ var_1 ... var_N) :id my_label)
...
(assert (= total (+ my_label ...)))
```

or by using one of the available *C API* functions, e.g.

```
msat_term internal_term = msat_objective_get_term(env, obj);
```

The requirement for an explicit declaration and the reification of objectives into terms, which includes MAXSMT goals, allows for combining an objective function with other terms and objectives appearing in the same formula to create novel constraints or to compose novel objectives. Given an arbitrary set of objectives $\{\mathsf{obj}_1, ..., \mathsf{obj}_N\}$, this can be done by linear composition if every obj_i is (convertible to) $\mathcal{LIRA}$, or via a MAXMIN/MINMAX composition (§4.6.1) when all objectives have the same type.

5.3 Input/Output Interfaces

The functionality of OPTIMATHSAT can be accessed directly, using its *command line* interface, or through its public API. This section includes the following content:

§5.3.1 The *Extended* SMT-LIBV2 *Interface*, accessible through the *command line*.

§5.3.2 The MINIZINC *Interface*, accessible through the *command line*.

§5.3.3 The *Public API* of OPTIMATHSAT, accessible through its library.

The primary goal of the material presented here is to introduce the interest reader to the usage of OPTIMATHSAT. Similarly to a tutorial, the presentation includes at least one usage example for each interface. A secondary goal, in particular for the *Extended* SMT-LIBV2 *Interface*, is to provide some context and justification for certain design choices that were made in the definition of some language extensions.

Regardless of the interface being used, an understanding of the the rich set of *configurable options* of OPTIMATHSAT can be helpful to squeeze the best performance out of the tool. For

this reason, we separately describe these options, common for all input interfaces, in Section §5.4.

5.3.1 Extended SMT-LIBv2 Interface

One of the cornerstones of SMT is represented by the SMT-LIB initiative [smt], that develops and promotes a standardized *command-line* input interface for all SMT solvers. In contrast, the current state of OMT solving is a much more fragmented reality, due to the lack of a uniform syntax that is both accepted by all OMT solvers and also handled in the same way. In an effort to guarantee the maximum level of interoperability among solvers, and possibly move towards a de facto standard, the input/output interface of OPTIMATHSAT has been gradually updated over the years to become as compatible as possible with that of other OMT solvers, e.g., Z3. Even so, there are still important and irreconcilable differences in the input/output interface of OPTIMATHSAT that make it unique and require additional design care and translation effort when modeling problems for multiple OMT solvers. This is a a consequence of the *compositional approach* adopted by OPTIMATHSAT, described in Section §5.2, as well as of other design choices that depend on the underlying SMT solver, MATHSAT5.

In the following, we provide a comprehensive list of the syntactic extensions to the SMT-LIBv2 standard introduced by OPTIMATHSAT to deal with *Optimization Modulo Theories* instances. Then, we illustrate a running example taken from [ST18] that uses this interface, and conclude with a discussion about the remaining compatibility issues among OPTIMATHSAT and Z3 using the same running example.

Extended SMT-LIBv2 Syntax

Hereafter, we use square brackets to highlight optional syntactic elements of a syntactic extension that can be omitted when not needed. Moreover, we use `<const_term` to denote a term, of the same type of the objective function, with a constant value.

Objectives Declaration. OPTIMATHSAT provides two essential ways to declare, and simultaneously push on the objectives stack, an optimization goal.

```
(minimize <term> [:id <string>] [:signed]
            [:lower <const_term>] [:upper <const_term>])
(maximize <term> [:id <string>] [:signed]
            [:lower <const_term>] [:upper <const_term>])
```

The command `minimize` pushes the $\mathcal{LIRA}$, $\mathcal{BV}$ or $\mathcal{FP}$ `<term>` on the internal stack of objectives, so that it can be minimized at the next `(check-sat)` call. Dual for `maximize`.

```
(minmax <term> ... <term> [:id <string>] [:signed]
            [:lower <const_term>] [:upper <const_term>])
(maxmin <term> ... <term> [:id <string>] [:signed]
            [:lower <const_term>] [:upper <const_term>])
```

The syntactic sugar extension `minmax` pushes on the objectives stack a fresh `<term>`, of the same type as the terms `<term> ... <term>`, whose optimum value matches with the minimum maximum value of the argument list of terms `<term> ... <term>` (see Section §4.6.1). Dual for `maxmin`.

Both types of objective declarations admit optional attributes that serve one of the following two purposes.

- **Objective Naming.** The attribute `:id` is used to assign an explicit name to an objective function. To associate the customized *alias* to the objective function, the former must be declared of the same (or compatible) type of the latter. Naming an objective has two useful applications. First, it can be used to retrieve the value of the objective function from the satisfiable model, using the standard SMT-LIBv2 command `(get-value (<string>))`. Second, it can be used to compose objective functions with one another, as described in §5.2.

  Remark 5.3.1. The SMT-LIBv2 standard provides the command `define-fun`, that can be used to achieve similar goals as the `:id` attribute. Therefore, the `:id` attribute does not increase the overall expressiveness of the language. Nonetheless, we deemed useful extending the optimization syntax for objectives declaration with this attribute for consistency with the case of $\mathcal{PB}$/MAXSMT goals defined in terms of soft clauses using the command `assert-soft`. A secondary advantage of this extension is that it simplifies, at the implementation level, the identification of any occurrence of objective recombination. In the future, this may be leveraged with new techniques that exploit this knowledge.

- **Sign Specification.** In the case of a $\mathcal{BV}$ objective, the `:signed` attribute can be used to tell OPTIMATHSAT that the value of the objective function should be interpreted as a *signed Bit-Vector* rather than as an *unsigned Bit-Vector*, which is the default interpretation. Currently, the `:signed` attribute is ignored when the objective function is of a different type other than Bit-Vector.

- **Binary/Adaptive Search.** Either (or both) of the optional attributes `:lower` and `:upper` can be added to the definition of an objective function to restrict its feasible domain. In the case of a `minimize` or `maxmin` command, the lower bound is considered not strict, whereas the upper bound is considered strict. The interpretation is dual for `maximize` and `minmax`.

  The non-strict bound is required by OPTIMATHSAT to run the optimization search with the *binary-* and *adaptive-search* modes described in Section §2.3.1. In absence of a non-strict bound, OPTIMATHSAT runs in *linear-search* mode by default, regardless of the tool's configuration. OPTIMATHSAT is able to automatically determine the lower and upper bound of an objective, without any input from the end-user, only when dealing with $\mathcal{PB}$/MAXSMT objectives solely defined in terms of the `assert-soft` command.

  We remark that, in the case of the Multiple-Independent (a.k.a. Boxed) Optimization approach described in Section §4.6.2, using either of these attributes to impose a bound over an objective obj_i does not affect the feasible domain of other objectives in the same formula, even when these have one or more variables in common with one another (e.g. x and $2x$). This is due to the definition of Multiple-Independent Optimization (see §4.6.2), that requires the OMT solver to treat each objective obj_i as a completely independent unit from the other goals in the same formula. In all other cases, bounding the objective function is akin to pushing a bounding constraint on the formula stack, that affects the feasible space of the formula as a whole, at least up until when the objective declaration does not get out of scope.

$\mathcal{PB}$/MAXSMT Terms. In OPTIMATHSAT, Pseudo-Boolean and MAXSMT goals and constraints should be defined in terms of soft clauses using the following command.

```
(assert-soft <term> [:id <string>] [:weight <const_term>])
```

The extension `assert-soft` adds `term` on the stack of soft clauses with a weight, of $\mathcal{LIRA}$ type, equal to `<const_term>` (1 if omitted). All soft clauses with the same `:id` belong to the same $\mathcal{PB}$/MAXSMT term; if a soft clause is not assigned an `:id` attribute, then it is added to the default group with *label* I.

The `assert-soft` command can be used to encode a *Generalized* MAXSMT instance directly (see Section §2.3.3), i.e. the extension admits soft clauses with zero or negative weights. This design choice allows for easiness of use, in particular when the command is used to express general Pseudo-Boolean objectives and constraints. Internally, OPTIMATHSAT automatically transforms the input set of soft clauses into a new set of purely-positive weighted soft

clauses, as described in Section §2.3.3. This transformation step is required for compatibility with the MAXSAT engines of OPTIMATHSAT (like, e.g., the MAXRES engine described in Section §4.2.2 and any other external MAXSAT engine that can be plugged into OPTIMATHSAT using the lemma-lifting approach presented in [CGSS13a]), which commonly operate under the assumption of a positive-weighted set of soft clauses.

Given a positive-weighted set of soft clauses φ_s belonging to the same id group, OPTIMATHSAT constrains id to be equal to $\sum_i \text{ITE}(C_i, 0, w_i)$, where C_i is a soft clause with weight w_i, and ITE is a function returning 0 when C_i is assigned to true and w_i otherwise. In OPTIMATHSAT, the resulting id term can be arbitrarily minimized, maximized[18], used to express *cardinality constraints* or composed with other $\mathcal{LIRA}$, $\mathcal{BV}$, $\mathcal{FP}$, $\mathcal{PB}$ or MAXSMT objectives and terms to build mixed Boolean/numeric objective functions as in [TSP17, NSGM16a].

Remark 5.3.2. We recall that this is in contrast with what is currently possible in Z3, which always implicitly defines a minimization objective for each group of soft clauses, thus forbidding any form of objective combination when this language construct is used. For more details, see Section §5.2.

Objectives Stack. The SMT-LIBV2 standard defines three commands for managing the assertion stack of an incremental SMT solver, namely `push`, `pop` and `reset-assertions`. To deal with OMT, OPTIMATHSAT extends the definition of these commands as follows.

```
(push <numeral>)
```

Given a `<numeral>` of value n, pushes n empty objective levels on the objectives stack. If n is equal to 0, no objective levels are pushed.

```
(pop <numeral>)
```

Given a `<numeral>` of value n, smaller than the number of objective levels in the stack, pops the n most-recent objective levels from the stack of objectives. If n is equal to 0, no objective levels are popped. The first objective level, which is not created by a push command, cannot be popped.

```
(reset-assertions)
```

[18] This can be useful, for example, when the starting set of soft clauses contains negative-weighted soft clauses. The end-user is thus relieved from the necessity of manually encoding the problem under the more restrictive positive-only requirement of the usual notion of a MAXSMT problem.

Removes from the objective stack all objective levels beyond the first one. In addition, it removes all objectives from the first objective level.

Optimization Search. The following SMT-LIBV2 command, that instructs an SMT solver to check for the satisfiability of the conjunction of all formulas in the current context, is extended by OPTIMATHSAT as follows.

```
(check-sat)
```

If the stack of objectives is empty, the OMT solver behaves as a regular SMT solver. Otherwise, OPTIMATHSAT solves the corresponding *Optimization Modulo Theories* instance. When the solver finishes attempting to do this, it replies on its regular output channel according to the multi-objective combination approach selected in the configuration[19]. In Multiple-Independent (§4.6.2) and Lexicographic (§4.6.3) optimization, OPTIMATHSAT answers with SAT if it found a satisfiable solution for at least one objective function, with UNSAT if no solution was found for the input formula, and with UNKNOWN otherwise. In incremental Pareto (§4.6.4) optimization, OPTIMATHSAT answers with SAT each time it finds a Pareto-optimal solution, with UNSAT when it exhausted the (possibly empty) set of Pareto-optimal solutions and with UNKNOWN otherwise. The incremental Pareto-optimization search is restarted after an UNSAT result or a command that alters the state of either the assertion or the objective stacks.

As described in Section §4.7, OPTIMATHSAT can enumerate all possible satisfiable assignments to a fixed set of Boolean variables (i.e. `<const_term> ... <const_term>`) that make a single-objective or a lexicographic OMT problem optimal. The following SMT-LIBV2 extension, which is also available in MATHSAT5 when dealing with a purely SMT instance, is used to access this functionality.

```
(check-allsat (<const_term> ... <const_term>))
```

In the case of a satisfiable input formula, the output of this command extends the one of a regular `(check-sat)` call by enumerating all of the satisfiable assignments to the given list of interesting Boolean variables.

Inspecting Models. OPTIMATHSAT introduces two new extensions for inspecting the optimal model(s) of an OMT formula.

[19]Currently, the various OMT solvers apply different multi-objective combination approaches by default. Therefore, it is preferable to always explicitly configure this option in the OMT formula. We refer to Section §5.4.6 for instructions on how to do so.

```
(get-objectives)
```

This command prints, on the regular output channel, the value of all objective functions currently stored in the stack of objectives. In the case that an objective function has not yet been optimized, or OPTIMATHSAT was unable to determine the satisfiability of the input formula before ending the search, the printed value is equal to UNKNOWN. In Lexicographic and Pareto optimization mode, OPTIMATHSAT the printed value is equal to UNSAT under the same conditions that make `(check-sat)` print the same value. Instead, in Multi-Independent (a.k.a Boxed) optimization, the printed value in correspondence with an objective obj_i is equal to UNSAT when the conjunction of the input formula with the bounds imposed on the declaration of the objective obj_i is unsatisfiable. This means that, in this optimization mode, there can be some unsatisfiable objectives even when the input formula is satisfiable.

When the optimization search is configured to terminate prematurely, OPTIMATHSAT also prints the latest search interval for any objective that was not fully optimized.

```
(load-objective-model <numeral>)
```

Given a positive `<numeral>` of value n, this command loads in the OMT solver's environment the satisfiable model corresponding to the objective having index n in the internal stack of objectives. The internal stack of objectives is indexed starting from zero, so that the index 0 always corresponds to the least recently declared objective among those still on the stack. In the case of a negative `<numeral>` of value m, OPTIMATHSAT accesses the objectives stack in reverse order. For instance, the index -1 always corresponds to the most recently declared objective on the stack, and the index -2 maps to the second-last objective declaration, if any.

In single-objective, Lexicographic and Pareto-optimization modes, OPTIMATHSAT loads the optimum model in the environment, whenever available, without any explicit request from the end-user. Instead, in the case of a multi-objective instance executed in Multiple-Independent optimization mode, it is necessary to use this command to make the optimum model accessible for inspection.

Extended SMT-LIBv2 Example

We use a simple OMT problem as an excuse to illustrate some language features of the extended SMT-LIBv2 syntax introduce by OPTIMATHSAT with a practical example. We note that the same example has been previously presented in [ST18], in which we used an earlier version of the tool that printed a different output trace from the current version.

Example 5.3.1. *A 3D printing company must print* 1100 *units of the latest trending 3D model for an urgent delivery that must be handled within a single day. To perform this task, it can use four printers* M_0, M_1, M_2, M_3*. Given the desired 3D model, the estimated maximum daily production capacity of each machine is of* 800 *units for* M_0*,* 500 *units for* M_1*,* 600 *units for* M_2 *and* 200 *units for* M_3*. Taking into consideration the fixed electricity cost, the average maintenance cost of each machine and the per-unit filament cost, the company estimates that —for each unit being produced— the operating cost for machines* M_0, M_1, M_2 *and* M_3 *is equal to* $8, 9, 9$ *and* 5 *euro respectively.*

The manager of this company is tasked with the goal of solving two different problems for this delivery. A first goal is to find a production allocation of the machines such that (A) the overall production cost is minimized and, at a tie, (B) the least number of machines is used. A second, alternative, goal is to find a production allocation that minimizes the total cost (C), which is given by the sum of the production cost with the compensation for the employees handling the task. For this goal, the manager knows that (I) each machine needs to be supervised by a different technician, (II) each technician is compensated with a daily wage equal to 32.5 *euro and that (III) if a machine remains unused for the whole day, the technician does not need to show up at work and the manager is not required to provide for any compensation.*

Figure 5.3 shows an encoding of this simple OMT problem with the extended SMT-LIBv2 *syntax. The lines from* 1 *to* 24 *encode the part of the problem that is common for both goals. The lines from* 25 *to* 35 *encode the first goal as a lexicographic OMT instance over the two objectives (A) and (B). Last, the lines from* 36 *to* 42 *encode the second, alternative, goal as a single-objective OMT instance with (C) as the only target function.*

The formula in Figure 5.3 is solved by OPTIMATHSAT *in negligible time. The output of the OMT solver is shown in Figure 5.4.*

The lexicographic-optimal solution for the first goal, shown at lines 2-5*, reveals that the minimum production cost is equal to* 8300 *euro and that to print* 1100 *units of the desired item within a day it is necessary to use at least* 3 *machines. Due to the symmetry among machines* M_1 *and* M_2*, there are several ways to allocate the production so that it meets the demand without exceeding the same minimum production cost of* 8300 *euro. This is reflected in the model found by* OPTIMATHSAT *when solving for (A), shown at lines* 6-15*, which uses both* M_1 *and* M_2 *under their maximum capacity, printing* 99 *units with the first machine and only* 1 *with the latter. The tie is broken by solving for (B) after fixing the optimum value of target (A). The corresponding model, shown at lines* 16-25*, shows a better allocation of the resources used for this task that allows machine* M_2 *to not be used.*

The optimal solution for the second goal (C), which optimizes the total cost for processing the order, is shown at lines 27-30 *and it corresponds to* 8397.50 *euro.* ◇

```
 1  (set-option :produce-models true)  ; enable print model
 2  (declare-fun production_cost () Real)
 3  (declare-fun q0 () Int)            ; machine 'i' production load
 4  (declare-fun q1 () Int)
 5  (declare-fun q2 () Int)
 6  (declare-fun q3 () Int)
 7  (declare-fun m0 () Bool)           ; machine 'i' is used
 8  (declare-fun m1 () Bool)
 9  (declare-fun m2 () Bool)
10  (declare-fun m3 () Bool)
11  (assert (<= 1100 (+ q0 q1 q2 q3))) ; set goods quantity
12  (assert (and                       ; set goods produced per machine
13      (and (<= 0 q0) (<= q0 800)) (and (<= 0 q1) (<= q1 500))
14      (and (<= 0 q2) (<= q2 600)) (and (<= 0 q3) (<= q3 200))
15  ))
16  (assert (and                       ; set machine `used` flag
17      (=> (< 0 q0) m0) (=> (< 0 q1) m1)
18      (=> (< 0 q2) m2) (=> (< 0 q3) m3)
19  ))
20  (assert (= production_cost (+ (* q0 8) (* q1 9) (* q2 9) (* q3 5)) ))
21  (assert-soft (not m0) :id used_machines)
22  (assert-soft (not m1) :id used_machines)
23  (assert-soft (not m2) :id used_machines)
24  (assert-soft (not m3) :id used_machines)
25  (push 1)
26  (minimize production_cost)
27  (minimize used_machines)
28  (set-option :opt.priority lex)
29  (check-sat)                        ; optimize (A), (B) lexicogr.
30  (get-objectives)
31  (load-objective-model 0)           ; print model for (A)
32  (get-model)
33  (load-objective-model 1)           ; print model for (B) after (A)
34  (get-model)
35  (pop 1)
36  (minimize (+ production_cost (* (/ 325 10) used_machines))
37            :id total_cost)
38  (set-option :opt.priority box)
39  (check-sat)                        ; optimize only (C)
40  (get-objectives)
41  (load-objective-model 0)           ; print value of (C)
42  (get-value (total_cost))
```

Figure 5.3: Extended SMT-LIBv2 example (Formula taken from [ST18]).

```
 1  sat
 2
 3  (objectives
 4   (production_cost 8300)
 5   (used_machines 3)
 6  )
 7  ( (production_cost 8300)
 8    (q0 800)
 9    (q1 99)
10    (q2 1)
11    (q3 200)
12    (m0 true)
13    (m1 true)
14    (m2 true)
15    (m3 true)
16    (used_machines 4) )
17  ( (production_cost 8300)
18    (q0 800)
19    (q1 100)
20    (q2 0)
21    (q3 200)
22    (m0 true)
23    (m1 true)
24    (m2 false)
25    (m3 true)
26    (used_machines 3) )
27  sat
28
29  (objectives
30   ((+ production_cost (* (/ 65 2) used_machines)) (/ 16795 2))
31  )
32  ( (total_cost (/ 16795 2)) )
```

Figure 5.4: Extended SMT-LIBv2 example output.

Extended SMT-LIBv2 Compatibility with Z3

The extended SMT-LIBv2 encoding shown in Example 5.3.1 is not compatible with Z3, for the following reasons:

(i) The formula explicitly requests the minimization of the Pseudo-Boolean sum corresponding to the MAXSMT group with identifier `used_machines` (line 27). In Z3, this results in an error because a PB sum defined in this way is always implicitly minimized.

(ii) When Z3 assigns an implicit minimization goal to a MAXSMT group, it does so at a location that corresponds to the `assert-soft` definition. This means that, given the lexicographic OMT formula displayed in Figure 5.3, Z3 assigns a higher priority to the minimization of `used_machines` than to the minimization of $production\_cost$, thus searching for an entirely different lexicographically-optimal solution than OPTIMATHSAT.

(iii) The definition of goal (C) at the lines 36 and 37 of Figure 5.3 leverages the *compositional approach* of OPTIMATHSAT, described in Section 5.2, to express $total\_cost$ as the linear combination of a $\mathcal{LIRA}$ term and a MAXSMT objective. This definition is currently illegal in Z3 because it does not admit any use of the identifier of a group of soft clauses.

(iv) The label placed on goal (C) is not recognized by the parser of Z3.

(v) The extension `load-objective-model`, used to retrieve various optimum models in the given example, is not supported by Z3.

Naturally, it is possible to find an alternative formulation of the problem that is both accepted and interpreted in the same manner by both OPTIMATHSAT and Z3. However, for considerably larger instances, this might negatively affect the performance of OPTIMATHSAT.

5.3.2 MINIZINC Interface

As part of the work of this Ph.D., OPTIMATHSAT was extended with a new input/output interface for dealing with *Finite Domain Constraint Programming* (FDCP) problems encoded in MINIZINC, the most widely adopted language in the field of FDCP solving that we succinctly introduced in Section §2.4.2.

FLATZINC Handling. As mentioned in Section §2.4.2, a MINIZINC model is typically flattened into a FLATZINC [fla] instance, using the MZN2FZNcompiler, before being handed over to a MINIZINC solver. The purpose of FLATZINC is to bridge the gap among the high-level modeling in MINIZINC, and the need for a fixed, and easy-to-parse, input format that simplifies the implementation of the input interface of a MINIZINC solver.

The MINIZINC Interface of OPTIMATHSAT targets the version 1.6 of the FLATZINC standard, that includes the most consolidated, and used, features of the language. Currently, OPTIMATHSAT supports most constraints defined in this version of the standard, excluding only those that require the ability to use trigonometric, power or logarithmic functions and those that require non-linear arithmetic reasoning in general[20]. All annotations in the FLATZINC model are currently ignored, except for those marking output variables and arrays, that are used to identify the interesting variables of the problem that have to be included in the printed model.

The *global constraints* in MINIZINC express more complex relations among the objects of the language than the regular FLATZINC constraints. Normally, a MINIZINC solver is not required to directly support any *global constraints*, as these can be compiled by the MZN2FZN tool into a standardized FLATZINC representation that uses only regular constraints and, if necessary, a number of fresh support variables. Even so, it can be convenient for a MINIZINC solver to handle *global constraints* directly, especially when it can use *ad hoc* decision procedures for dealing with them efficiently. At the moment, OPTIMATHSAT does not implement specialized decision procedures for any *global constraints*. Nonetheless, it has some special support for a few *global constraints*, that consists in a suitable internal representation that leverages OPTIMATHSAT capabilities as efficiently as possible. For instance, this allows OPTIMATHSAT to use *sorting networks* to directly encode *cardinality constraints*, that are otherwise expressed in terms of $\mathcal{PB}$ and $\mathcal{LIRA}$ constraints when going through the MZN2FZN compiler.

Internally, OPTIMATHSAT represents the three basic types of FLATZINC as follows. A FLATZINC `bool` is mapped into a *Boolean*. A FLATZINC `int` is represented using either the theory of *Linear Integer Arithmetic* ($\mathcal{LIA}$) or the theory of *Bit-Vectors*, depending on the tool's configuration. Last, a FLATZINC `float` is represented with the theory of *Linear Rational Arithmetic* ($\mathcal{LRA}$), despite the fact that OPTIMATHSAT inherits from MATHSAT5 efficient decision procedures for dealing with the theory of *Floating-Point* such as the *Abstract CDCL* engine presented in [HGBK12]. Although an encoding of this type was not explored so far, and it may allow for an easy representation of some of the FLATZINC language features that

[20] An extended list of all the currently supported constraints is available at the web-page [opt]. We plan to extend OPTIMATHSAT to support the complete FLATZINC 1.6 standard. This requires the completion of the ongoing integration process with the new decision procedures of MATHSAT5 for transcendental and non-linear arithmetic, that were recently introduced in the SMT solver [CGI+18].

are currently still not being supported, using the theory of $\mathcal{LRA}$ seems to provide an easier integration with constraints using the theory of $\mathcal{LIA}$.

By default OPTIMATHSAT eliminates, whenever possible[21], any 0-1 Integer variable contained in the FLATZINC model. If successful, the 0-1 Integer variable is then subsequently replaced with its *Boolean* counterpart inside any constraint referencing it. This transformation is meant to improve OPTIMATHSAT's overall performance, as performing Boolean Constraint Propagation is typically much cheaper than using its decision procedure for *Mixed Linear Integer and Rational Arithmetic* ($\mathcal{LIRA}$).

The only two non-basic types that are made available in FLATZINC are internally handled by OPTIMATHSAT as follows. A FLATZINC `set` is encoded with an occurrence representation approach that uses additional Boolean variables as witnesses of existence within the set, similarly to [Ach09]. A FLATZINC `array`, instead, is given no explicit representation because it is a simple container for other, related, objects and there is no need for any reasoning capability over it, like the one offered by the theory of *Arrays* in SMT.

Multi-Objective FLATZINC Extension. The FLATZINC parser in OPTIMATHSAT deviates from the official specification of the language in a fundamental aspect. In fact, it allows for multiple objective functions to be defined within the same model, even though this is not allowed by the standard.

When the problem contains multiple objectives, these should be specified in a comma-separated list contained in the same `solve` constraint. For instance, in the following snippet of an extended FLATZINC model, OPTIMATHSAT is requested to minimize `goal_1` and maximize `goal_2`.

```
solve minimize goal_1, maximize goal_2;
```

The solution of a multi-objective FLATZINC model depends on the multi-objective optimization approach being used, that can be chosen among the Multiple-Independent (a.k.a. Boxed), the Lexicographic and the Pareto optimization modes described in Section §4.6. The desired multi-objective combination mode can be selected using the command-line configurable option `-opt.priority=[box|lex|par]`, as described in Section §5.4.6.

MINIZINC to FLATZINC Conversion. A MINIZINC model must first be converted in the FLATZINC format before OPTIMATHSAT can solve it. This task requires the MZN2FZN com-

[21] Typically, this transformation is effective only when BOOL2INT constraints appear as early as possible in the input model.

piler, that is distributed in the same package as the MINIZINC software library[22]. The simplest way to use the MZN2FZN compiler is without any options, except for the name of the output file. This results in a FLATZINC model that can be parsed by any MINIZINC solver, as it contains only those constraints that are defined in the FLATZINC standard.

```
$ mzn2fzn model_file.mzn [data_file.dzn] \
          -o output_file.fzn
```

Alternatively, it is possible to generate a FLATZINC model that specifically targets OPTIMATHSAT, by instructing the MZN2FZN compiler to not translate those *global constraints* for which the OMT tool has special support.

If the library of *global constraints* that belongs to OPTIMATHSAT, called `smt2`, has already been installed on the system, it is sufficient to invoke the MZN2FZN tool as follows.

```
$ mzn2fzn -G smt2 \
          model_file.mzn [data_file.dzn] \
          -o output_file.fzn
```

Otherwise, if the directory of global constraints `smt2` is missing, this can be installed as follows. First, download the `smt2.tar.gz` package from the website of OPTIMATHSAT[23]. Second, unpack the package inside the directory of global constraints of the target MINIZINC distribution[24].

FLATZINC to SMT-LIBV2 Conversion. An interesting aspect of OPTIMATHSAT is that it can be used not only as a MINIZINC solver, but also as a converter from one file format to the other. In particular, it can convert a FLATZINC model into an OMT instance encoded with the *Extended* SMT-LIBV2 *Syntax* described in Section §5.3.1[25].

[22] Alternatively, when dealing with MINIZINC models containing Floating-Point values, we reccomend using the EMZN2FZN compiler made available at the address `https://github.com/PatrickTrentin88/emzn2fzn`. The tool is designed to avoid rounding of Floating-Point values when flattening the input model.

[23] Direct link: `http://optimathsat.disi.unitn.it/data/smt2.tar.gz`.

[24] For versions 1.∗ of the MZN2FZN tool, the directory of global constraints is located at the path `${MINIZINC_PATH}/lib/minizinc`. For version 2.0 and newer, the directory has been moved to the path `${MINIZINC_PATH}/share/minizinc`.

[25] An alternative is represented by the FZN2SMT compiler published at [fzn]. We note, however, that FZN2SMT converts a model in the SMT1 file format, with no optimization extension, while OPTIMATHSAT uses the *Extended* SMT-LIBV2 format that specifically aims to OMT solvers.

To perform this conversion, we exploit the *API tracing* functionality inherited from MATHSAT5. The following command produces an OMT instance encoded with the same internal representation used by OPTIMATHSAT to handle the input FLATZINC model.

```
$ optimathsat -input=fzn \
              -debug.api_call_trace=1 \
              -debug.api_call_trace_dump_config=False \
              -debug.solver_enabled=False \
              -debug.api_call_trace_filename=output.smt2 \
              < input.fzn
```

In the case that the resulting OMT instance needs to be handed over to some other OMT solver that does not interpret the `assert-soft` extension in the same way as OPTIMATHSAT (like, e.g., Z3), it is necessary to add the option `-opt.debug.expand_soft=True` to the previous command. As described in Section §5.4.1, this eradicates the compatibility issue by replacing any group of soft clauses with an adequate Pseudo-Boolean encoding in the resulting OMT formula.

To solve the generated OMT instance with OPTIMATHSAT, the OMT solver should be executed with a few extra options, as in the following example. This ensures that OPTIMATHSAT is run with the same configuration used when dealing with FLATZINC models.

```
$ optimathsat -model_generation=true \
              -opt.print_objectives=true \
              -opt.par.mode=callback \
              < output.smt2
```

Extended FLATZINC Example.

Hereafter, we provide step-by-step instructions on how to use OPTIMATHSAT to solve a MINIZINC instance of of the well-known NP-hard Cutstock problem, which goal is to minimize the amount of stock material used to produce a number of goods.

Example 5.3.2. *The* MINIZINC *model for the Cutstock problem, shown in Figure 5.5, is taken from the the official* MINIZINC *distribution [Minb][26]. Constraint* `C.1`*, at lines* 14-16*, ensures that the number of pieces being produced is sufficient to cover the demand for each type of good. Constraint* `C.2`*, at lines* 18-21*, ensures that total amount of stock material being used does not*

[26]For the purposes of this example, we performed some minor change to the original model.

```
%------------------------------------------------------------------------%
% Jakob Puchinger <jakobp@cs.mu.oz.au>, December 2007                    %
%------------------------------------------------------------------------%

int: L;                                 % stock unit length
int: K;                                 % max no. of stock units used
int: N;                                 % number of pieces
array[1..N] of int: lengths;            % pieces length
array[1..N] of int: demands;            % pieces demand
array[1..K] of var 0..1: pieces;        % 1: stock used, 0: otherwise
array[1..K, 1..N] of var 0..K: items; % pieces cut from stock unit k
var int: obj;                           % objective

constraint forall(i in 1..N) (                                    % C.1
               sum([ items[k, i] | k in 1..K ]) >= demands[i]
           );

constraint forall( k in 1..K ) (                                  % C.2
               sum(i in 1..N) (items[k,i] * lengths[i])
               <= pieces[k] * L
           );

constraint obj = sum([ pieces[k] | k in 1..K ]);                  % C.3

solve minimize obj;

output [ "Cost = ",  show( obj ), "\n" ] ++
       [ "Pieces = \n\t" ] ++ [show(pieces)] ++ [ "\n" ] ++
       [ "Items = \n\t" ] ++
       [ show(items[k, i]) ++ if k = K then "\n\t" else " " endif |
         i in 1..N, k in 1..K ] ++ [ "\n" ];

% data
N = 3;
L = 10;
K = sum(demands);
lengths = [7, 5, 3];
demands = [2, 2, 4];
```

Figure 5.5: MINIZINC Cutstock example.

exceed the available resources. Last, constraint `C.3`*, at line* 23*, sets the objective function* obj *to be equal to the total number of stock material units used.*

The MINIZINC *model is compiled into* FLATZINC *with the command*

```
$ mzn2fzn -G smt2 cutstock.mzn
```

that results in the FLATZINC *instance of the Cutstock problem shown in Figure 5.6. This simple instance of the Cutstock problem is solved by* OPTIMATHSAT *in negligible time. As witnessed by the following output trace, the optimal value of the objective function is* 4*.*

```
$ optimathst -input=fzn < cutstock.fzn

% objective: obj (optimal model)
obj = 4;
items = array2d(1..8, 1..3, [0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
                             0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1]);
pieces = array1d(1..8, [0, 0, 1, 0, 0, 1, 1, 1]);
----------
==========
```

Experimental Evaluation. In Section §6.6, we show an experimental evaluation using this (novel) MINIZINC Interface and comparing the performance of OPTIMATHSAT against those of other Finite Domain Constraint Programming tools on formulas taken from the official MINIZINC competition and benchmark-sets steaming from the domain of Formal Verification.

5.3.3 API Interface

OPTIMATHSAT extends the *C API* of MATHSAT5 with the new *Optimization Modulo Theories* features. Like MATHSAT5, OPTIMATHSAT is distributed with scripts for building the *Python API*[27], using SWIG, and with a *Java API*. Both are automatically generated on top of the *C API*.

Introduction to the *C API*

In the following, we describe the most important *C API* extensions introduced by OPTIMATHSAT, with the goal of helping a new user getting started with the OMT library. For this purpose,

[27] The address `https://github.com/PatrickTrentin88/omt_python_examples` contains several usage examples of the *Python API* of OPTIMATHSAT.

```
 1  array [1..3] of int: demands = [2, 2, 4];
 2  array [1..3] of int: lengths = [7, 5, 3];
 3  array [1..24] of var 0..8: items :: output_array([1..8, 1..3]);
 4  var 0..8: obj :: output_var;
 5  array [1..8] of var 0..1: pieces :: output_array([1..8]);
 6  constraint int_lin_eq([-1, 1, 1, 1, 1, 1, 1, 1, 1], [obj, pieces[1],
       pieces[2], pieces[3], pieces[4], pieces[5], pieces[6], pieces[7],
       pieces[8]], 0) :: defines_var(obj);
 7  constraint int_lin_le([7, 5, 3, -10], [items[1], items[2], items[3],
       pieces[1]], 0);
 8  constraint int_lin_le([7, 5, 3, -10], [items[4], items[5], items[6],
       pieces[2]], 0);
 9  constraint int_lin_le([7, 5, 3, -10], [items[7], items[8], items[9],
       pieces[3]], 0);
10  constraint int_lin_le([7, 5, 3, -10], [items[10], items[11], items
       [12], pieces[4]], 0);
11  constraint int_lin_le([7, 5, 3, -10], [items[13], items[14], items
       [15], pieces[5]], 0);
12  constraint int_lin_le([7, 5, 3, -10], [items[16], items[17], items
       [18], pieces[6]], 0);
13  constraint int_lin_le([7, 5, 3, -10], [items[19], items[20], items
       [21], pieces[7]], 0);
14  constraint int_lin_le([7, 5, 3, -10], [items[22], items[23], items
       [24], pieces[8]], 0);
15  constraint int_lin_le([-1, -1, -1, -1, -1, -1, -1, -1], [items[1],
       items[4], items[7], items[10], items[13], items[16], items[19],
       items[22]], -2);
16  constraint int_lin_le([-1, -1, -1, -1, -1, -1, -1, -1], [items[2],
       items[5], items[8], items[11], items[14], items[17], items[20],
       items[23]], -2);
17  constraint int_lin_le([-1, -1, -1, -1, -1, -1, -1, -1], [items[3],
       items[6], items[9], items[12], items[15], items[18], items[21],
       items[24]], -4);
18  solve minimize obj;
```

Figure 5.6: FLATZINC Cutstock example.

we assume that the reader is at least a bit familiar with the library of MATHSAT5. We refer the interested reader to the official MATHSAT5 and OPTIMATHSAT websites, [matb] and [opt], for more detailed documentation of the *C API*.

Environment. The *Optimization Modulo Theories* functionality of OPTIMATHSAT is accessible through an environment instance apt for optimization.

```
msat_env msat_create_opt_env(msat_config cfg);
void     msat_destroy_env(msat_env e);
```

The function `msat_create_opt_env()` creates an instance of environment with access to all OMT functionality. The argument of type `msat_config` is a container that, like in MATHSAT5, allows the end-user to provide their own configuration overriding the default behavior of the OMT solver. The returned value type is `msat_env`, which is the same used for an environment instance by MATHSAT5. As a consequence, the environment for optimization can also be used as a regular MATHSAT5 environment, and it gives access to the same *Satisfiability Modulo Theories* functionalities. When the environment is no longer used, it should be manually destroyed with the function `msat_destroy_env()` to properly free all of its resources.

Objectives Declaration. A simple minimization or maximization objective can be created using one of the following commands.

```
msat_objective msat_make_minimize(msat_env e, msat_term goal,
                                  msat_term lower, msat_term upper);
msat_objective msat_make_maximize(msat_env e, msat_term goal,
                                  msat_term lower, msat_term upper);
```

Both functions take as argument the instance of (optimizing) environment in which the goal should be created (`env`), an expression representing the objective function (`goal`), plus (optional) lower and upper bound values for the optimization search. When the lower bound is equal to `msat_error_term`[28], the goal is assumed to not be lower-bounded. Dual for the upper bound.

The following commands can be used to declare either a MINMAX or a MAXMIN goal, described in Section §4.6.1.

[28]This value can be instantiated with the MATHSAT5 function `msat_make_error_term()`.

```
msat_objective msat_make_minmax(msat_env e, size_t l, msat_term ts[],
                                msat_term lower, msat_term upper);
msat_objective msat_make_maxmin(msat_env e, size_t l, msat_term ts[],
                                msat_term lower, msat_term upper);
```

Both functions take as input the reference (optimizing) environment in which the objective is going to be created (`env`), the number of terms contained in the MINMAX/MAXMIN objective (`l`), the corresponding list of terms to be used upon creating the MINMAX/MAXMIN objective (`ts[]`), plus (optional) lower and upper bound values for the optimization search. The terms contained in the list `ts[]` must be all of the same (or compatible) type. The bounds can be omitted using the same approach described for a simple objective declaration.

Altogether, these four functions allow for declaring arbitrary $\mathcal{LIRA}$, *unsigned* $\mathcal{BV}$, $\mathcal{FP}$, Pseudo-Boolean and MAXSMT objectives. In addition, for each of these commands OPTIMATHSAT provides an alternative variant that can be used to declare a *signed* $\mathcal{BV}$ objective. The suffix "`_signed`" is appended to the name of each function, while the argument list remains unchanged. When declaring an objective of any type other than $\mathcal{BV}$, the original command and its variant result in the same objective instance being created.

After an objective has been declared, it is possible to use the following *macro* to check that it has been correctly instantiated.

```
int MSAT_ERROR_OBJECTIVE(msat_objective o);
```

The function returns `1` if the `msat_objective` variable points to a valid objective and `0` otherwise.

When an objective is no longer needed, its allocated resources should be manually freed using the following function.

```
void msat_destroy_objective(msat_objective o);
```

We note that, in the case that the objective has been previously pushed on the objectives stack and it has not yet gone out of scope, destroying an objective does not result in an immediate deallocation of all of its resources. The goal instance will be automatically destroyed after it is removed from the stack of objectives.

Objectives Stack. In contrast with what happens when using the *Extended* SMT-LIBV2 *syntax* (see Section §5.3.1), OPTIMATHSAT does not automatically push each new objective on

the stack of objectives when using the *API*. Instead, the objective should be manually pushed on the stack of objectives with the following command.

```
int msat_assert_objective(msat_env e, msat_objective o);
```

The stack of objectives is handled in parallel with the stack of formulas, in the same fashion described for the *Extended* SMT-LIBV2 *Interface* in Section §5.3.1.

```
int msat_push_backtrack_point(msat_env e);
int msat_pop_backtrack_point(msat_env e);
int msat_reset_env(msat_env e);
```

The function `msat_push_backtrack_point()` pushes a checkpoint for backtracking in an environment, while the function `msat_pop_backtrack_point()` backtracks to the last checkpoint set in an environment. The function `msat_reset_env()`, instead, clears both the assertion stack and the objectives stack. Any objective that has not yet been manually destroyed remains valid even after being removed from the objectives stack, so that it can be used multiple times.

OPTIMATHSAT provides some functionality for iterating over the stack of objectives that are currently asserted, to facilitate keeping track of the objective functions loaded in the (optimizing) environment. The function

```
msat_objective_iterator msat_create_objective_iterator(msat_env e);
```

creates an objective stack iterator for the given environment. When

```
int msat_objective_iterator_has_next(msat_objective_iterator i);
```

returns a value different from 0, then

```
int msat_objective_iterator_next(msat_objective_iterator i,
                                 msat_objective *o);
```

saves in the pointer `*o` the next goal in the stack of objectives. Whenever an objective iterator is no longer needed, its resources should be released by invoking

```
void msat_destroy_objective_iterator(msat_objective_iterator i);
```

$\mathcal{PB}$/MAXSMT Terms. As illustrated in Section §5.3.1 for the *Extended* SMT-LIBV2 *Interface*, in OPTIMATHSAT, Pseudo-Boolean and MAXSMT goals and constraints should be defined in terms of soft clauses. The experimental evaluations in Section §6.2 witness that this approach can significantly improve the performance of OPTIMATHSAT because it enables the use of more sophisticated encodings and techniques at solving time. The command

```
int msat_assert_soft_formula(msat_env e, msat_term cl, msat_term w
                                                    const char *id);
```

adds the soft clause `cl` with weight `w` to the (possibly empty) set of soft clauses with the identifier `id`. The weight `w` must be a constant value of $\mathcal{LIRA}$ type and it can be negative, zero or positive. This allows for a direct encoding of *Generalized* MAXSMT instances (see Section §2.3.3). Internally, OPTIMATHSAT automatically transforms the input set of soft clauses into a new set of purely-positive weighted soft clauses, as described in Section §2.3.3. This transformation, completely transparent to the end-user, is necessary for compatibility with the MAXSAT engines that can be used to deal with the OMT instance. We recall that OPTIMATHSAT uses MAXRES (see Section §4.2.2) as internal MAXSAT engine, and that any external MAXSAT engine can be plugged into OPTIMATHSAT using the lemma-lifting approach presented in [CGSS13a].

Optimization Search. An *Optimization Modulo Theories* instance is solved by invoking the following command.

```
msat_result msat_solve(msat_env e);
```

When the stack of objectives is empty, OPTIMATHSAT behaves like a regular SMT solver. Otherwise, the OMT problem is solved according to the selected configuration of the tool. In Multiple-Independent (§4.6.2) and Lexicographic (§4.6.3) optimization, the value returned by this function is equal to SAT if a satisfiable solution was found for at least one objective function, it is equal to UNSAT if OPTIMATHSAT was able to determine the unsatisfiability of the OMT instance and it is equal to UNKNOWN otherwise. In incremental Pareto (§4.6.4) optimization, the function returns SAT each time a new Pareto-optimal solution is found, it returns UNSAT when the (possibly empty) set of Pareto-optimal solutions is exhausted and it returns UNKNOWN otherwise. The incremental Pareto-optimization algorithm is restarted after each UNSAT result or after a call to a function that alters the state of the assertion or the objective stacks.

In addition to the `msat_result` value returned by solving an OMT instance, which refers

to the optimization search as a whole, OPTIMATHSAT separately associates to each objective function its own `msat_opt_result` value that can be retrieved with the following function.

```
msat_opt_result msat_objective_result(msat_env e, msat_objective o);
```

Depending on the status of the optimization search on the objective specified in the argument list, three possible values can be returned: SAT indicates that OPTIMATHSAT found some solution for the objective function, UNSAT indicates that the objective function is unsatisfiable with respect to the input formula and UNKNOWN is used when the OMT solver is unable to decide the satisfiability. The OMT solver further distinguishes a SAT result in three possible cases: OPTIMAL, meaning that OPTIMATHSAT was able to find (and certify) the optimal solution, APPROX, meaning that the optimization search was interrupted due to meeting the early termination criteria specified in the configuration, and PARTIAL, that is used for all other causes of early termination of the optimization search (like, e.g., a timeout). In the case of a APPROX or PARTIAL result, the model stored by OPTIMATHSAT in the objective function may or may not be optimal.

Inspecting Models. When *model generation* is enabled, OPTIMATHSAT generates a satisfiable model of the input formula each time it updates the upper [resp. lower] bound of some objective function being minimized [resp. maximized]. OPTIMATHSAT provides two ways to access the model associated with an objective function. The function

```
int msat_load_objective_model(msat_env e, msat_objective o);
```

replaces the current model of OPTIMATHSAT with the model stored in the optimization goal, whereas

```
msat_model msat_get_model(msat_env e, msat_objective o);
```

returns the model of the objective function directly.

In both cases, the content of the model can be inspected using the *model iteration* and *evaluation* functionalities made available by MATHSAT5. When the model instance is no longer needed, this should be properly deallocated with

```
void msat_destroy_model(msat_model m);
```

In general, the model stored in an objective function is not guaranteed to be optimal, unless the corresponding `msat_opt_result` is equal to SAT and also OPTIMAL. In the case of an unbounded or infinitesimal $\mathcal{LIRA}$ objective, OPTIMATHSAT generates a model in which the value of the objective function is given a finite representation determined by the configuration of the tool (see Section §5.4.4).

Other Functionalities. We refer to the official website [opt] for a detailed description of the remaining, less prominent, functionalities provided by the *API* of OPTIMATHSAT. These include

- Methods for retrieving additional status information at the end of the optimization search. This is useful, for example, to extract the final lower and upper bound and evaluate the overall progress when the optimization search is early terminated.
- Methods for performing a *callback-based* exploration of the Pareto front.
- The porting of the lemma-lifting interface presented in [CGSS13a] into OPTIMATHSAT, that allows one to use an external MAXSAT solver to deal with OMT($\mathcal{PB} \cup \mathcal{T}$)/MAXSMT instances.

and more.

C API Example

Hereafter, we illustrate, with a short code example, a possible use of the *C API* of OPTIMATHSAT for solving the same OMT problem illustrated in the Example 5.3.1. A very similar example was presented in [ST18], that, however, used an earlier version of OPTIMATHSAT and a less updated version of the *API*.

Example 5.3.3. *Figure 5.7 shows the relevant parts*[29] *of a possible* C API *encoding of the OMT problem described in Example 5.3.1.*

The target problem is a Lexicographic *OMT instance. By default,* OPTIMATHSAT *handles multiple objectives according to the* Multiple-Independent *(a.k.a. Boxed) OMT combination. Therefore, the code at lines* 6-9 *overrides the default configuration of* OPTIMATHSAT*, enabling both lexicographic optimization (line* 7*) and model generation (line* 8*). Then, the configuration is used to create an* optimizing *instance of the environment, including all of the OMT*

[29] In this example, we omit the part of the source code that is solely based on functionalities shared with the underlying SMT solver MATHSAT5, and focus instead on the new optimization features introduced with OPTIMATHSAT. The complete source code is made available on the official website of the OMT solver, [opt], listed among the other *API* examples.

```
 1  #include "mathsat.h"
 2  #include "optimathsat.h"
 3
 4  int main(int argc, char *argv[])
 5  {
 6      msat_config cfg = msat_create_config();     /* 1. configuration */
 7      msat_set_option(cfg, "opt.priority", "lex");
 8      msat_set_option(cfg, "model_generation", "true");
 9      ...
10
11      msat_env env = msat_create_opt_env(cfg);    /* 2. environment   */
12
13      msat_term formula;                          /* 3. formula       */
14      ...
15
16      msat_assert_formula(env, formula);
17
18      msat_term production_cost = ... ;           /* 4. objectives    */
19
20      msat_assert_soft_formula(env, not_m0, one, "used_machines");
21      ...
22
23      msat_term used_machines = msat_from_string(env, "used_machines");
24      ...
25
26      msat_objective obj[2];
27      obj[0] = msat_make_minimize(env, production_cost, none, none);
28      obj[1] = msat_make_minimize(env, used_machines, none, none);
29
30      msat_assert_objective(env, obj[0]);         /* 5. optimization  */
31      msat_assert_objective(env, obj[1]);
32
33      msat_result res = msat_solve(env);
34
35      msat_load_objective_model(env, obj[1]);     /* 6. dump model    */
36      ...
37
38      msat_destroy_objective(env, obj[1]);        /* 7. free memory   */
39      msat_destroy_objective(env, obj[0]);
40      msat_destroy_env(env);
41      msat_destroy_config(cfg);
42  }
```

Figure 5.7: Sample *C API* code.

functionalities, at line 11. *We note that, as described earlier in this Section,* OPTIMATHSAT *uses distinct functions to instantiate an* optimizing *and a* non-optimizing *version of the environment. The code at lines* 13-16*, which is almost completely omitted as it uses only functionalities shared with* MATHSAT5*, corresponds to the* SMT-LIBV2 *constraints at lines* 11-19 *of Figure 5.3. As in Example 5.3.1, the first objective is a $\mathcal{LIRA}$ expression labeled with the name* `production_cost` *and the second goal is a $\mathcal{PB}$ term encoded as a group of soft clauses with identifier the* `used_machines` *(lines* 20-24*). The pair of minimization objectives is then created at lines* 26-28*. We note that the* `used_machines` *term used at line* 28 *is instantiated by calling the* MATHSAT5 *function* `msat_from_string()` *over the identifier of the* MAXSMT *sum after the group of soft clauses has been created. Similarly to the* Extended SMT-LIBV2 *example, no lower and upper bound is imposed on either objective. Hence, the* empty *term "*`none`*", created with* `MSAT_MAKE_ERROR_TERM()`*, is used as placeholder for the mandatory objective bounds. The resulting* Lexicographic OMT *problem is then solved at line* 33*. By default,* OPTIMATHSAT *automatically sets the optimum model of a* Lexicographic OMT *instance as the current model of the solver, so that it can then be accessed in the same way as in* MATHSAT5*. However, since this is not necessarily the case for every multi-objective combination approach, we show how to explicitly ask the OMT solver to do so at line* 35*. Following the best practice, the resources that are no longer used are manually released at lines* 38-41.

The complete version of the source code show in Figure 5.7, available on OPTIMATHSAT*'s website, is then compiled and executed as follows.*

```
$ g++ example_api.cpp \
      -I${OPTIMATHSAT_DIR}/include/ \
      -L${OPTIMATHSAT_DIR}/lib/ \
      -lmathsat -lgmp -lgmpxx -lstdc++ \
      -o example_api
$ ./example_api
Lex. optimum: 8300, 3
```

5.4 Configurable Options

After several years of research and development, OPTIMATHSAT now implements a broad number of techniques, that can be fine-tuned using its rich set of configurable options. Whenever necessary, we describe each configurable option in detail, to allow for an educated choice

when using OPTIMATHSAT. We observe that, depending on the interface that is being used to interact with the tool, there are several ways to set a configurable option.

- The option is specified as a command-line argument. Some options, like `-input=...` and `-config=...`, should only be used as command-line arguments.

```
~$ optimathsat -option=VALUE ...
```

- The option is stored in a newline-separated configuration file, that is then separately handed over to OPTIMATHSAT with the command-line option `-config=FILE`. The arguments of the options should be in lowercase format, unless differently specified by the `-help` page.

```
option=VALUE
...
```

- The option is added directly to the input formula using the SMT-LIBV2 syntax. Options should be preferably listed in the header part of a formula, both because it can be too late to set some of these options the OMT started to process the problem, and also because this increases the visibility of any hard-coded option that might override any command-line argument.

```
(set-option :config option=VALUE)
...
```

- When using OPTIMATHSAT via its public *API*, the configuration is specified with the following function, same as MATHSAT5.

```
msat_set_option(cfg, "option", "VALUE")
```

5.4.1 Input/Output Interface Options

The default experience with OPTIMATHSAT can be personalized via a set of configurable options that allow the end-user to adjust the input/output behavior of the tool.

Input format. When the command-line interface is used, it is possible to specify the language in which the input formula is encoded with the option

```
-input=STR                                                    [default: smt2]
```

As seen in §5.3, OMT problems can be encoded using either one of the following two formats: `smt2`, that corresponds to the *Extended* SMT-LIBV2 *syntax* for OMT (see Section §5.3.1), and `fzn`, that accepts formulas adhering to the FLATZINC standard, plus minor language extensions introduced by OPTIMATHSAT to support multi-objective combinations (see Section §5.3.2).

Verbosity. Another, often useful, option is

```
-opt.verbose=BOOL                                             [default: false]
```

When this flag is enabled, OPTIMATHSAT displays additional status info along the optimization search. In particular, it signals when the lower or the upper bounds of some objective function gets updated by the optimization search.

Tracing. Among the rich set of features that are inherited for free from MATHSAT5, the *API Tracing* functionality deserves a special mention. Although its main purpose is for logging reasons, it can also be used to automatically generate OMT instances in the *Extended* SMT-LIBV2 *syntax* starting from a FLATZINC problem, as seen in §5.3.2. By default, the resulting instance is generated with an encoding that is best-suitable for OPTIMATHSAT, so that it can be solved as efficiently as possible. However, the same formula might not be correctly handled when handed over to Z3, due to a diverging handling of the `assert-soft` command. OPTIMATHSAT provides the following option to overcome this issue.

```
-opt.debug.expand_soft=BOOL                                   [default: false]
```

When the flag is enabled, any soft clause is automatically expanded into a Pseudo-Boolean encoding. Otherwise, the `assert-soft` command is used.

Retro-compatibility. Prior to version 1.5.0, OPTIMATHSAT printed by default the values of all objective functions when used from the command-line. Since then, in compliance with the recent changes introduced in Z3, the *Extended* SMT-LIBV2 *syntax* has been enriched with the command `(get-objectives)`, that is used to issue an explicit request to display these

values. As a result, newer versions of OPTIMATHSAT no longer autonomously display such information by default. The following option was added to the tool for retro-compatibility.

```
-opt.print_objectives=BOOL                                    [default: false]
```

When this option is enabled and OPTIMATHSAT is used from the command-line, the solver prints the value of all objective functions after each `(check-sat)`, even without an explicit call to the new `(get-objectives)` command.

Since version 1.5.0 onward, the output format used by OPTIMATHSAT has also been updated to be more uniform with that of Z3, in an effort to move forward towards a more easily inter-operable environment. We introduced the following option, that controls the output format used by the solver, for retro-compatibility.

```
-opt.output_format=STR                                        [default: new]
```

The parameter can be set to `old`, that refers to the output format used by OPTIMATHSAT up to version 1.4.5, or to `new`, that is the new standard adopted from version 1.5.0 onward.

5.4.2 OMT-based Search Options

Search Strategies. As described in §2.3.1, in OPTIMATHSAT the OMT-based search advances towards the optimum goal either in *linear-*, *binary-* or *adaptive-search* mode. The strategy can be selected with the option

```
-opt.strategy=STR                                             [default: lin]
```

The value `lin` is used for *linear-search*, `bin` for *binary-search* and `ada` for *adaptive-search*. We recall that, in general, it is necessary to provide an initial *lower bound* [resp. *upper bound*] when minimizing [resp. maximizing] some mixed $\mathcal{LRIA}$, $\mathcal{PB}$ or MAXSMT objective function to use the *binary-* and *adaptive-search* strategies. This information is not strictly necessary when dealing with some $\mathcal{BV}$ or $\mathcal{FP}$ goal, although it can speedup the search.

When OPTIMATHSAT is executed in *binary-* or in *adaptive-search* mode, the optimization search can be further adjusted using one of the following configurable options.

```
-opt.bin.pivot_position=FLOAT                                 [default: 0.5]
```

This option controls the desired aggressiveness of the *pivoting* cuts that are generated along the optimization search in *binary-* and *adaptive-search* mode (see Section §2.3.1). A valid argument is contained in the interval $]0, 1]$, and it indicates the relative position of the *pivoting* value with respect to to the range of values that goes from the *lower bound* up to the *upper bound*. For example, the value 0.5 corresponds to bisecting the search interval in two halves of the same size. In general, the most suitable value to be assigned to this option depends on various factors —including, e.g., the tightness of the initial range of the objective function that is provided by the end-user, the encoding of the formula and the problem itself— and it is therefore best determined hands-on.

```
-opt.bin.first_step_linear=BOOL                    [default: true]
```

This flag forces the first search step to run in *linear-search* mode. This is useful to obtain a tighter estimate of the initial *upper bound* [resp. *lower bound*] when minimizing [resp. maximizing] some objective function. We note that, in OPTIMATHSAT, a tighter initial search interval can positively affect the effectiveness of *pivoting*, at least for the first few steps.

```
-opt.bin.max_consecutive=INT                       [default: 1]
```

This option limits the maximum number of consecutive *pivoting* steps before OPTIMATHSAT is forced to perform a *linear-search* step. As seen in §2.3.1, this is especially useful when dealing with $\mathcal{LIRA}$/$\mathcal{LRA}$ objectives, because it prevents 'Zeno-ness" behavior when the optimization search is advanced.

Search Learning. As described in §4.5, an *incremental* OMT solver is able to exploit learned information across multiple optimization searches to improve the overall performance. In this regard, OPTIMATHSAT can be optionally configured to learn, along the optimization search, additional trivial implications which aim is to increase the chance that learned information is re-used in subsequent *incremental* calls to the OMT solver.

```
-opt.learn_trivial_implications=BOOL               [default: true]
```

When this flag is enabled, OPTIMATHSAT learns $\mathcal{T}$-valid clauses of the form $(\mathsf{obj}_i < u_i) \rightarrow (\mathsf{obj}_i < u'_i)$, where u_i is the most recently found minimum value of obj_i and u'_i is the previous value of u_i (Dual for maximization). As a result, as soon as the literal corresponding to $(\mathsf{obj}_i < u_i)$ is assigned to true in a subsequent *incremental* call to the OMT solver, then all previously-

learned clauses in the form $\neg(\mathsf{obj}_i < u_i') \vee C$ are "activated". For more details, we refer the interested reader to [ST15c].

5.4.3 Early Termination and Approximate Search Options

Generally speaking, solving an OMT problem requires considerably more effort than dealing with the corresponding SMT problem obtained by discarding the objective functions. This has two main causes. The first is that, in contrast with an SMT solver that stops at the first solution, an OMT solver normally visits a (possibly large) number of intermediate solutions along the optimization search. The second reason is that an OMT solver must *certify* the optimality of the latest solution it finds before it is allowed to terminate the optimization search. In practical terms, this means searching for another improving solution and proving that it does not exist. The experimental evidence of [ST12, ST15a, ST15c] on OMT($\mathcal{LIRA} \cup \mathcal{T}$) formulas has shown that, on the benchmark-sets being subject to investigation, the *certification* step can be quite expensive in practice and require up to (about) half of the entire search-time.

In some particular resource-demanding OMT applications a "good enough" estimate of the optimal solution might be preferred over a *certified* optimal solution when the former can be found in a relatively short amount of time compared to the latter. For example, this can be the case for time-sensitive applications in which there is a trade-off in-between the desirable progress in the optimization of some objective function and the amount of time that can be devoted to this task.

For this reason, OPTIMATHSAT provides a number of useful options that allow for terminating the search early and/or searching for an approximate solution.

Early Termination. The easiest way to end the optimization search early is to use the following option, that forces the OMT-based optimization search described in §2.3.1 to stop as soon as every objective function being tracked has at least one (possibly not optimal) satisfiable solution.

```
-opt.no_optimization=BOOL                                [default: false]
```

We notice that enabling this flag forces OPTIMATHSAT to behave similarly to a standard SMT solver, which would also stop at the first solution, with the significant difference that OPTIMATHSAT performs an additional call to the $\mathcal{T}$-minimization procedure for each objective in the input formula. The returned model, is thus guaranteed to be minimal for the given $\langle \mu, \mathsf{obj} \rangle$ pair, where μ is the first (complete) satisfiable truth assignment that was found by OPTIMATHSAT and obj is the objective function (Dual in maximization). The main use-case of this option

is to allow for designing new optimization procedures on top of OPTIMATHSAT—via its public *API*— that need to have access to the underlying $\mathcal{T}$-minimization procedures but do not want to rely on the inline optimization-search procedure described in §2.3.1.

When dealing with time-sensitive applications, instead, the typical scenario involves the use of some type of software timeout.

```
-timeout=FLOAT                                              [default: 0]
```

This option can be used to impose a timeout, expressed in milliseconds, on the running-time of the solver. We notice that the timer starts ticking from the moment in which the OMT search is started and that a value equal to 0 means that no timeout is set.

```
-opt.soft_timeout=BOOL                                      [default: false]
```

If this flag is enabled, the timeout set with `-timeout=FLOAT` is ignored if it occurs before when each objective function has at least one (possibly suboptimal) satisfiable solution. If this is the case, then the search is automatically terminated as soon as the latter condition becomes is verified. In essence, the idea is to postpone an otherwise hard timeout that could terminate the search before the solver is able to determine the satisfiability of the input formula. The use-case scenario for this option is represented by those OMT applications for which an approximated estimate of the optimal solution is acceptable, but the absence of any solution is not (see, e.g., [TSP17]).

Approximate Search. In some OMT applications, the time required to solve a problem is not the only factor to take into consideration. Instead, another equally important property to be considered is the perceived quality of a solution, often expressed in terms of the amount of (estimated) progress made by the OMT search.

To this aim, OPTIMATHSAT provides two heuristic functions that can be used to perform an approximated optimization search.

```
-opt.abort_interval=FLOAT                                   [default: 0.0]
```

When the "abort interval" is set to a value v larger than 0.0, the optimization search stops as soon as the size of the search interval becomes smaller than the threshold value v.

```
-opt.abort_tolerance=FLOAT                                  [default: 0.0]
```

When the "abort tolerance" is set to a value r larger than 0, the optimization search stops as soon as the ratio among the current search interval size and the initial search interval size becomes smaller than the value r.

We note that, in the case of Multiple-Independent optimization, the requirement for terminating the whole optimization search is that the search interval of each objective being tracked meets the condition for termination. Up until when every objective satisfies such condition, the OMT solver is allowed to find an improving solution for any of them.

5.4.4 $\mathcal{T}$-Specific Options

General $\mathcal{T}$-Optimization. When minimizing an objective obj_i, OPTIMATHSAT invokes a $\mathcal{T}$-specific minimization procedure over each satisfiable and complete truth assignment μ that propositionally satisfies the input formula, provided that this function is available for the theory $\mathcal{T}$ (see Section §2.3.1). This behavior can be globally adjusted with the following option.

```
-opt.theory.no_optimization=BOOL                    [default: false]
```

Toggling this flag disables the call to the $\mathcal{T}$-minimization procedure. Without calling this function, for each complete truth assignment μ that satisfies the input formula, OPTIMATHSAT is forced to generate an arbitrary model that does not necessarily make the objective function minimal for the given truth assignment μ. As a consequence, the optimization search can be forced to iterate over the same truth assignment μ multiple times, with a (generally) negative impact on the performance. Overall, the optimization search may still be able to eventually converge at the optimal solution when dealing with some theory $\mathcal{T}$ and some objective obj_i for which there can only be finitely many enumerable solutions. However, the progress towards the optimum solution can be much slower and in the case of $\mathcal{LIRA}$ objectives termination is not guaranteed.

$\mathcal{LIRA}$-Theory

Search Behaviour. The OMT-based optimization procedures for OMT ($\mathcal{LRA}$) and OMT ($\mathcal{LIRA}$), described in §2.3.1 and §4.1 respectively, can be fine-tuned with any combination of the following options.

```
-opt.theory.la.ignore_non_improving=BOOL            [default: true]
```

When this flag is enabled, the $\mathcal{T}$-minimization procedure for $\mathcal{LIRA}$ objectives is invoked only when the current truth assignment μ admits an improving solution for a given objective obj_i. In

single-objective optimization, this condition is always verified so the value assigned to the option is ignored. Instead, when dealing with Multiple-Independent (a.k.a. Boxed) optimization, it can happen that the set of $\mathcal{T}$-atoms handed over to the $\mathcal{T}$-minimization procedure does not admit any improving solution for some objective obj_i (but not all). In this situation, OPTIMATHSAT performs an additional $\mathcal{T}$-satisfiability check over the set of constraints $\mu \wedge (\mathsf{obj}_i < \mathsf{ub}_i)$, where ub_i is the most recently found upper bound of obj_i. Whenever the set of constraints is found to be $\mathcal{T}$-inconsistent, OPTIMATHSAT can thus avoid wasting time by calling the (usually more expensive) $\mathcal{T}$-minimization procedure.

```
-opt.theory.la.lar_always_optimize=BOOL                [default: false]
```

In single-objective optimization, enabling this option guarantees that the value of the objective function is always optimal at the end of each satisfiability check inside the Simplex-based tableau embedded in the $\mathcal{LRA}$-Solver. In other words, OPTIMATHSAT invokes the $\mathcal{T}$-minimization procedure after each satisfiability check, even on partial truth assignments. While this feature may improve the performance when dealing with OMT problems with a strong theory-combination component, it causes also some additional overhead. In multi-objective optimization, this option has no effects.

As described in Section §4.1, the Branch&Bound search embedded in the $\mathcal{T}$-minimization procedure for $\mathcal{LIRA}$ objectives comes in two variants.

```
-opt.theory.la.laz_mode=STR                                [default: part]
```

When set to `part`, OPTIMATHSAT uses the "truncated" version of the $\mathcal{LIA}$-minimization procedure, and when set to `full`, OPTIMATHSAT runs the complete, but possibly more expensive, variant of $\mathcal{LIRA}$-MINIMIZE(). On the one hand, the "truncated" version of the search can result in the same truth assignment μ being generated multiple times along the optimization search. On the other hand, OPTIMATHSAT is better equipped to handle those degenerate cases for which a simple Branch&Bound search is not powerful enough, because it can exploit the $\mathcal{LIA}$-Solver of MATHSAT5 more thoroughly.

Model Values. Although internally the OMT solver is capable of dealing with $\mathcal{LIRA}$ constraints using infinite precision and symbolic computation, OPTIMATHSAT does not yet support generating a *symbolic* model of the input formula. Instead, it uses some finite notion of "infinity" and "epsilon" to instantiate a *concrete* model of the input formula. Both of these parameters can be fine-tuned with the following options.

```
-opt.theory.la.infinite_pow=INT                                [default: 9]
```

With this option, the value of "infinity" is set to 10^N, where N is the (strictly) positive value used as an argument. When an objective obj_i is not lower-bounded, the model-construction routine guarantees that the corresponding $\min_{\mathsf{obj}}$ value is smaller or equal to -10^N. Dual for maximization.

```
-opt.theory.la.delta_pow=INT                                   [default: 6]
```

Passing a value N to this option results in an internal parameter, called δ, to be set to the value 10^{-N}. Let $\min_{\mathsf{obj}} \stackrel{\text{def}}{=} \langle r, e \rangle$, where r is the *rational* component of $\min_{\mathsf{obj}}$ and e is the *infinitesimal* component of $\min_{\mathsf{obj}}$. Then, the model-construction routine guarantees that the concrete value of $\min_{\mathsf{obj}}$ used in the model $\mathcal{M}$ is chosen to be in the interval $[r, r + \delta \cdot e[$ when $e > 0$, and in $]r + \delta \cdot e, r]$ otherwise.

$\mathcal{BV}$-Theory

Engine. In OPTIMATHSAT, the engine used for the optimization of signed/unsigned Bit-Vector objectives can be selected with the option

```
-opt.theory.bv.engine=STR                                      [default: obvbs]
```

This option accepts three possible values: `omt` for the OMT-based search described in Section §4.3.1, `obvwa` for the OBV-WA algorithm [NR16] described in Section §4.3.2 and `obvbs` for the OBV-BS algorithm [NR16] described in Section §4.3.3.

Search Behaviour. As described in Section §4.3, the $\mathcal{BV}$-optimization routines in OPTIMATHSAT can be fine-tuned with the following two enhancements.

```
-opt.theory.bv.branch_preference=BOOL                          [default: false]
```

Enables the "branching preference" enhancement described in §4.3. With some simplification, enabling this option guarantees that the CDCL engine gives priority to the unassigned bits of the objective function first, starting from the MSB down to the LST, when branching over a variable. Only when each bit of the objective function is assigned a value, either by BCP or through a decision, the CDCL engine is allowed to branch over other unassigned literals.

```
-opt.theory.bv.polarity=BOOL                                [default: true]
```

With some simplification, when this option is enabled, the phase-saving value of the bits of a $\mathcal{BV}$ objective is initialized to push the optimization search towards the direction of maximum gain. A more precise description is given in Section §4.3.

$\mathcal{FP}$-Theory

Engine. OPTIMATHSAT provides two engines for Floating-Point optimization, as described in Section §4.4, that can be selected with the following option.

```
-opt.theory.fp.engine=STR                                   [default: ofpbs]
```

The option can be set to be equal to either `omt`, `ofpbs` or `ofpbls`. The value `omt` corresponds to the OMT-based search described in Section §4.4.1. The OMT-based search is compatible with the *eager* approach for Floating-Point numbers, and also with the *lazy* approach when there is no constraint combining $\mathcal{BV}$ and $\mathcal{FP}$ terms together with one another, and the option `-theory.fp.bv_combination_enabled` can be set to false. The value `ofpbs` corresponds to the OFP-BS algorithm described in Section §4.4.2. This engine can handle $\mathcal{BV}$/$\mathcal{FP}$ combination, but it is only compatible with the *eager* Floating-Point engine. The value `ofpbls` runs a modified version of the OFP-BS algorithm described in Section §4.4.2 that encodes the problem without using $\mathcal{BV}$/$\mathcal{FP}$ combination. It is compatible with all three Floating-Point engines of OPTIMATHSAT: *eager*, *lazy* and *ACDCL*. Except for the case of the *eager* approach, it requires the option `-theory.fp.bv_combination_enabled` being set to false and it does not support $\mathcal{BV}$/$\mathcal{FP}$ combination. Furthermore, it is not compatible with any search enhancement.

Search Behaviour. The following search enhancements can be activated when dealing with some $\mathcal{FP}$ objective. Notice that the first two are the $\mathcal{FP}$-equivalent version of the corresponding $\mathcal{BV}$ options of the same name.

```
-opt.theory.fp.branch_preference=BOOL                       [default: false]
```

This option toggles the "branching preference" enhancement described in §4.4. With some simplification, setting it to true guarantees that the CDCL engine gives priority to the unassigned bits of the objective function first, starting from the MSB down to the LST, when branching over

a variable. Only when each bit of the objective function is assigned a value, either by BCP or through a decision, the CDCL engine is allowed to branch over other unassigned literals.

```
-opt.theory.fp.polarity=BOOL                                [default: true]
```

With some simplification, when this option is enabled, the phase-saving value of the bits of the $\mathcal{FP}$ objective is initialized to push the optimization search towards the direction of maximum gain. A more precise description is given in Section §4.4.

```
-opt.theory.fp.safe_bits_only=BOOL                          [default: true]
```

This flag controls the "safe bits restriction" enhancement described in Section §4.4. With some simplification, it restricts the scope of the other two enhancements so that they only affect those bits of the objective function for which the direction of maximum gain is certain at any given moment.

5.4.5 Cardinality Constraints Options

In regard to OMT($\mathcal{PB} \cup \mathcal{T}$) and MAXSMT solving, OPTIMATHSAT provides a comprehensive list of options that can be used to improve its performance. We recall that, at the time being, OPTIMATHSAT can take advantage from these advanced features only when the Pseudo-Boolean component of the input formula is encoded in terms of soft clauses, i.e. using the `assert-soft` command. In the future, we plan to extend OPTIMATHSAT to lift this limitation, e.g. by adding a preprocessing step with recognizes Pseudo-Boolean constraints while the formula is still being parsed.

Engine. Currently, there are three distinct engines for dealing with MAXSMT and Pseudo-Boolean objectives in OPTIMATHSAT. The engine can be selected using the following option.

```
-opt.maxsat_engine=STR                                      [default: omt]
```

The value `omt` corresponds to the usual OMT-based search described in Section §4.2.1. When the option is set to be equal to `maxres`, OPTIMATHSAT can use the MAXRES engine described in Section §4.2.2. Last, by setting the option to `ext`, OPTIMATHSAT can use an external MAXSAT engine to deal with the optimization problem. This feature is based on the so-called *Lemma Lifting* approach, briefly described in §2.3.3, presented by Cimatti et al. in [CGSS13a].

Encoding. As seen in Section §4.2.1, OPTIMATHSAT uses an *ad hoc* internal encoding for handling Pseudo-Boolean constraints specified with the `assert-soft` command. This encoding can be adjusted by means of any combination of the following options.

```
-opt.asoft.encoding=STR                                    [default: car]
```

This option selects the type of encoding used for Pseudo-Boolean and MAXSMT constraints. A value equal to `la` corresponds to the basic linear-arithmetic encoding described in Section §2.3.3. The values `seq` and `car` coincide with the *Bidirectional Sequential Counter Encoding* and the *Bidirectional Cardinality Network Encoding* respectively, both of which are described in Section §4.2.1.

```
-opt.asoft.circuit_limit=INT                                [default: 20]
```

When larger than zero, this option sets an upper bound on the maximum number of inputs that each (internally generated) sorting network circuit can have. In the case of Pseudo-Boolean terms that require more inputs than the given threshold to be represented, OPTIMATHSAT automatically splits these constraints into smaller and more easily manageable circuits. The goal of this option is to limit the amount of memory required to represent certain cardinality constraints. As shown in the experimental evaluation described in Section §6.2, this can be very effective at improving the performance of OPTIMATHSAT when the *Bidirectional Sequential Counter Encoding* is used.

```
-opt.asoft.reduce_vars=BOOL                                 [default: true]
```

When this flag is enabled, OPTIMATHSAT does not associate a fresh Boolean variable to each soft clause, as described in the encoding shown in Section §2.3.3. Instead, it uses the soft clause directly. Overall, this reduces the number of variables in the problem.

```
-opt.asoft.prefer_pbterms=BOOL                              [default: true]
```

When this flag is activated, the Boolean label associated with each soft clause is added to the list of preferred variables for branching, that is used by the internal CDCL engine to pick a new decision variable. Enabling this option typically improves the performance of the OMT solver. When the flag `opt.asoft.reduce_vars` is also enabled, then there is no Boolean label associated with a soft clause. In this case, this technique is only applied to those soft clauses whose guard is already a Boolean literal or its negation.

```
-opt.asoft.no_bidirection=BOOL                    [default: false]
```

When this option is used, OPTIMATHSAT uses an *asymmetric* version of the Pseudo-Boolean encoding described in §2.3.3 in which there is no bidirectional implication among a soft clause and its Boolean label. In this case, the identifier associated with each group of soft clauses should only be subject to upper-bounding and minimization. Any attempt to impose a lower bound or to maximize such identifier has no effect on the result of the optimization search. Discarding one direction of the implication relation can positively affect the search performance in some situations, but this should only be done when the input formula can allow it. This option is ineffective when `opt.asoft.reduce_vars` is enabled, because without an explicit labeling of soft clauses the encoding is always bidirectional.

5.4.6 Multi-Objective Options

Multi-Objective Combination. When the input formula contains multiple objectives, OPTIMATHSAT can handle it according to the various multi-objective combination approaches described in §4.6. Similarly to Z3, in OPTIMATHSAT the desired type of handling technique for multiple objectives can be selected with the following option.

```
-opt.priority=STR                                 [default: box]
```

Possible values are: `box` for the Multiple-Independent objective combination (a.k.a. Boxed) described in §4.6.2, `lex` for Lexicographic optimization (see Section §4.6.3) and `par` for Pareto optimization (see Section §4.6.4). The lack of an explicit option for the multi-objective MINMAX/MAXMIN combination described in Section §4.6.1 is intentional. In fact, as described in §5.3.1, OPTIMATHSAT uses a "clever" encoding to rewrite these problems into an instance of single-objective OMT.

In general, a good practice is to explicitly encode the desired value for this option directly inside the OMT formula, because different OMT solvers use different multi-objective combination approaches by default.

Lexicographic Optimization. OPTIMATHSAT provides two dedicated engines for lexicographic optimization that can be selected with the option

```
-opt.lex.engine=STR                               [default: ite]
```

Admissible values are: `uni` for the SMT-based UNIFIED_LEX algorithm and `ite` for the OMT-based ITERATED_LEX algorithm. Both algorithms are described in Section §4.6.3.

Pareto Optimization. OPTIMATHSAT is shipped with two engines for Pareto optimization. The option

```
-opt.par.engine=STR                                    [default: lex]
```

allows for an explicit selection of the desired Pareto-optimization engine. The parameter can be either `gia`, that corresponds to the basic *Guided Improvement Algorithm* implementation, or `lex`, that uses incremental Lexicographic optimization to extract Pareto-front solutions from the feasible space. A detailed description of both approaches is provided in Section §4.6.4.

The Pareto front of a multi-objective OMT problem can be explored either in *incremental* or *callback* mode.

In *incremental* mode, the execution of the Pareto-optimization algorithm is suspended whenever the OMT solver discovers a new Pareto-front solution. When this is the case, it returns the value SAT and it hands back execution control to the end-user, who has a chance to inspect the solver state and to retrieve the Pareto-optimal model. The Pareto-optimization algorithm is resumed on the next satisfiability check, and it terminates with UNSAT when the complete Pareto front has been explored, so that no other solution can be found. Notice that, when the Pareto front is explored incrementally, the search is automatically restarted after termination (i.e. after an UNSAT result), and also after any action that alters the state of either the stack of formulas or the objectives stack (i.e. after an `assert`, an `assert-soft`, a `push` or a `pop`).

In *callback* mode, upon receiving the request for a satisfiability check from the end-user, the Pareto-optimization algorithm is started and it is not interrupted up until when the entire Pareto front has been explored. The end-user must provide a callback function to inspect the solver state and to retrieve the Pareto-optimal solutions that are found during the search.

The following option, that accepts either `incremental` or `callback` as valid arguments, controls the Pareto-search mode.

```
-opt.par.mode=STR                                [default: incremental]
```

In the case that the end-user is interacting with OPTIMATHSAT from the command-line and that the OMT solver is running in `callback` mode, then it is possible to disable model-printing so that only the value of the objective functions is displayed on screen. This behavior is controlled by the option

```
-opt.par.print_model=BOOL                                    [default: true]
```

5.4.7 MINIZINC Options

When using the FLATZINC Interface of OPTIMATHSAT described in §5.3.2, the behavior of the OMT tool can be subject to further customization both in terms of Input/Output and also in terms of the internal representation that is given to the input model.

Search Exploration. In some FLATZINC applications, it is often useful to retrieve more than the optimal solution of the problem. For example, it might be useful to display all (possibly suboptimal) solutions that are encountered along the optimization search, or even all possible solutions of the input problem.

```
-opt.fzn.partial_solutions=BOOL                              [default: false]
```

When this flag is enabled, OPTIMATHSAT prints each (possibly suboptimal) model of the input problem that is encountered along the optimization search.

```
-opt.fzn.all_solutions=BOOL                                  [default: false]
```

If this option is enabled, OPTIMATHSAT does not stop at the first optimal solution that it finds, but it looks for other optimal models of the input formula. This concept is similar to the idea of "All-OMT" described in §4.7. Given a set of Boolean predicates $\mathcal{B}$, the goal of an All-OMT problem is to enumerate all possible assignments of values to $\mathcal{B}$ that satisfy the input formula and for which the objective function is fixed to its optimal value. In FLATZINC, the set $\mathcal{B}$ contains all output variables and output arrays of the input problem, that can be of any type, instead of just Boolean predicates.

In some situations, attempting to enumerate all optimal solutions can be an endeavoring, and ultimately unnecessary, task. In the case that the end-user is satisfied with a limited number of solutions, then the next option can be used to impose such an upper bound on the number of iterations performed by the routine for model-enumeration.

```
-opt.fzn.max_solutions=INT                                   [default: 0]
```

Encoding Variants. When a FLATZINC formula is parsed, the constraints in the input formula are transformed into a suitable internal representation that is based on the SMT-LIBV2 standard. Clearly, the performance of OPTIMATHSAT can be affected by this internal representation and yield different results depending on its effectiveness. Currently, various internal representations and heuristics are made available in OPTIMATHSAT through the following options.

```
-opt.fzn.ite_encoding=BOOL                                    [default: true]
```

Enabling this option activates an internal heuristic that eliminates 0-1 variables appearing in the original formula and substitutes their occurrences with a suitable propositional encoding. For the best results, all `bool2int` constraints in the input formula should appear at the top of the constraints section. Otherwise, OPTIMATHSAT can fail to recognize a 0-1 variable and eliminate it. The benefit of this heuristic is twofold. First, it reduces the computational effort imposed on the internal $\mathcal{LIRA}$-Solver, by shifting it on the more efficient Boolean Constraint Propagation engine. Second, it further improves the effectiveness of other heuristic enhancements (like, e.g., the next one in this list) when dealing with Pseudo-Boolean constraints.

```
-opt.fzn.asoft_encoding=BOOL                                  [default: true]
```

When this flag is enabled, Pseudo-Boolean constraints are heuristically mapped into a suitable internal encoding based on *soft clauses* rather than with the conventional linear programming encoding that uses the theory of $\mathcal{LIRA}$. This allows OPTIMATHSAT to use its sorting networks techniques, described in Section §4.2.1, and further improve the overall performance.

```
-opt.fzn.bv_all_different=BOOL                                 [default: true]
```

If this feature is enabled, the global constraint `all_different_int` is encoded using Bit-Vectors rather than $\mathcal{LIRA}$-constraints. This may improve the performance when dealing with a large number of elements or unsatisfiable combinations.

```
-opt.fzn.bv_integers=BOOL                                     [default: false]
```

When this option is enabled, OPTIMATHSAT uses the theory of Bit-Vectors rather than $\mathcal{LIA}$ to represent all Integer constants, variables and constraints that appear in the input problem.

Chapter 6

Experimental Evaluations

This part of the Ph.D. thesis is devoted to the experimental evaluations that were performed to investigate the behavior of OPTIMATHSAT and validate some of the technological advances in OMT described in §4.

The *Optimization Modulo Theories* features subject to our investigation are:

§6.1 The OMT($\mathcal{LIRA} \cup \mathcal{T}$) functionality described in §4.1 and restricted to the $\mathcal{LIA}$ case. The results of this experiment have been previously presented in [ST15c].

§6.2 The MAXRES engine and the sorting networks enhancement for dealing with OMT($\mathcal{PB} \cup \mathcal{T}$) and MAXSMT problems described in §4.2. The results of this experiments have been previously presented in [ST17].

§6.3 The procedures for OMT($\mathcal{BV} \cup \mathcal{T}$) described in §4.3. This is a new test and it does not appear in any previous publication.

§6.4 The procedures for OMT($\mathcal{FP} \cup \mathcal{T}$) described in §4.4. This is a new test and it does not appear in any previous publication.

§6.5 The incremental and multi-objective OMT features described in §4.5 and §4.6 respectively. These results have been first presented in [ST15c].

§6.6 A comparison among OMT and MINIZINC using OPTIMATHSAT as reference OMT tool. Part of the results presented here have been produced by F. Contaldo, a student at University of Trento, as part of their Master Thesis work.

The default *testing workbench* used for these experimental evaluations is a pair of two identical *8-core 2.20Ghz Xeon* machines with $64GB$ of RAM and running *Ubuntu Linux*. Unless differently specified, it can be assumed that each experiment was performed on this hardware configuration.

Note. We have collected together, in a single package, all the data related to the experimental evaluations presented in this Chapter, including the benchmark-sets, the scripts used to run the experiments, the tools being tested and the output data generated in the original experiment. We make this package publicly available at the address `http://disi.unitn.it/trentin/resources/phd_thesis_all.tar.gz`.

6.1 OMT ($\mathcal{LIA} \cup \mathcal{T}$) Evaluation

In the following, we summarize the salient points of an experimental evaluation over OMT($\mathcal{LIA} \cup \mathcal{T}$) that we have previously presented in [ST15c]. We refer the interested reader to that publication for more details on the experimental evaluation .

Experiment Setup. The benchmark-set is comprised by 544 OMT($\mathcal{LIA} \cup \mathcal{T}$) formulas derived from SMT-based Bounded Model Checking and K-Induction of parametric problems, that were generated with the SAL model checker[30]. The OMT tools considered in this experiment are OPTIMATHSAT (v. 1.1), Z3 (v. 4.3.3), and BCLT (v. 1.3). Since SYMBA does not support OMT($\mathcal{LIA} \cup \mathcal{T}$), this tool was not included in the experimental evaluation. Each solver was given up to $1200s$. to solve each OMT($\mathcal{LIA} \cup \mathcal{T}$) formula.

For what concerns OPTIMATHSAT, we evaluated its performance using the *linear-* (LIN), *binary-* (BIN) and *adaptive-search* (ADA) schemata described in §2.3.1. In addition, we tested the Branch&Bound optimization procedure of OPTIMATHSAT both with and without the "truncated" (TRN) enhancement described in §4.1.

Experiment Results. The results of this experiment are shown in table 6.1 and figure 6.1. Overall, both OPTIMATHSAT and Z3 were able to solve the whole benchmark-set, and Z3 used the least amount of time. The other tool under test, BCLT, had a timeout on 44 formulas.

When comparing the various OPTIMATHSAT configurations, we can make two observations. The first is that enabling the *adaptive-search* scheme, which uses an internal heuristic to dynamically choose between executing a *linear-* or a *binary-search* step, improves the performance of the OMT solver. The second observation is that using the "truncated" version of the Branch&Bound search does not result in a noticeable performance improvement. After taking a closer look to the set of benchmarks and to the output of the OMT solver, we conjecture that this is due to the fact that the optimization search is dominated by its Boolean component

[30] `http://sal.csl.sri.com/`.

tool, configuration & encoding	inst.	term.	timeout	time (s.)
BCLT	544	500	44	93040
Z3	544	544	0	**36089**
OPTIMATHSAT(LIN)	544	544	0	91032
OPTIMATHSAT(BIN)	544	544	0	99214
OPTIMATHSAT(ADA)	544	544	0	88750
OPTIMATHSAT(TRN+LIN)	544	544	0	91735
OPTIMATHSAT(TRN+BIN)	544	544	0	99556
OPTIMATHSAT(TRN+ADA)	544	544	0	88730

Table 6.1: A comparison on the performance of BCLT, Z3 and different configurations of OPTIMATHSAT on Bounded Model Checking problems.

when dealing with this particular benchmark-set. As a result, it is hardly likely that any enhancement on the optimization engine within the $\mathcal{LIA}$-Solver can have a positive impact on the performance of the OMT solver as a whole.

6.2 OMT ($\mathcal{PB} \cup \mathcal{T}$)/MAXSMT Evaluation

In the following, we provide an overview of an extensive experimental evaluation on OMT($\mathcal{PB} \cup \mathcal{T}$) and partial weighted MAXSMT problems that we have previously published in [ST17]. We refer the interested reader to this publication for more details on these experiments .

We categorize our experiments in two groups. The first group, including "CGMs with lexicographic OMT" (§6.2.1) and "CGMs with un-weighted MAXSMT" (§6.2.2), deals with problems that can be tackled with MAXSAT-based approaches like those described in §2.3.3 and §4.2.2. The second group, including "CGMs with MAXMIN OMT" (§6.2.3) and "LMT with OMT($\mathcal{PB} \cup \mathcal{T}$)" (§6.2.4), covers the case of OMT formulas in which the objective function has some non-$\mathcal{PB}$ component. For this reason, these cannot be handled with the more efficient MAXSAT-based approaches but only with the OMT-based techniques described in §2.3.3 and §4.2.1.

The main goals of these empirical evaluations are (1) to compare the performance of MAXSAT-based approaches with respect to OMT-based ones on those OMT problems in which techniques are applicable and (2) to evaluate the benefits of using sorting networks with OMT-based techniques as described in §4.2.1.

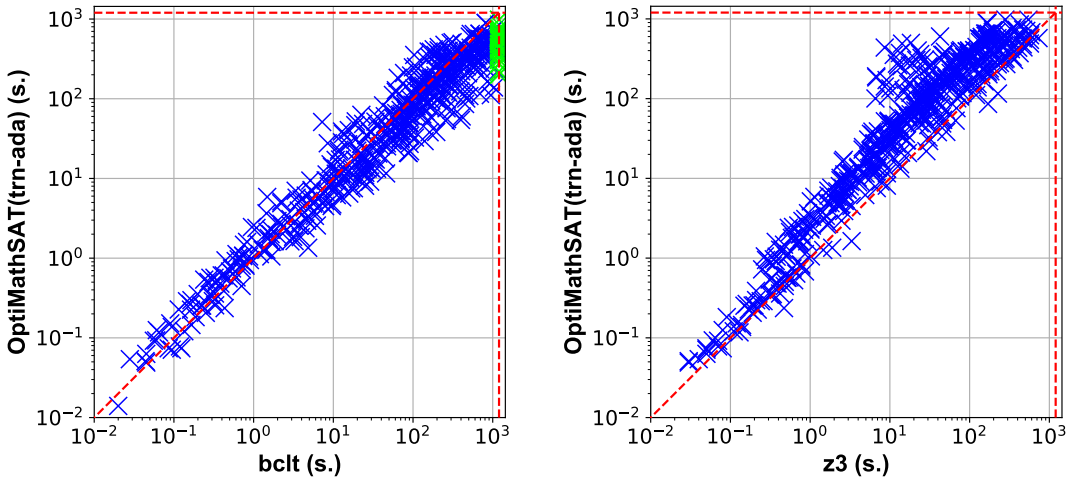


Figure 6.1: Pairwise comparisons between OPTIMATHSAT (TRN+ADA) and BCLT (left) and Z3 (right). (Blue points denote satisfiable benchmarks, green denotes a timeout.)

General Setup. The benchmark-sets used in this experimental evaluation on OMT($\mathcal{PB} \cup \mathcal{T}$) and MAXSMT were generated with one of the following two approaches. One group of formulas was generated with the CGM-TOOL [cgm], a modeling and automated-reasoning tool for requirement engineering, and it includes a number of OMT problems based on Constrained Goal Models [NSGM16b, NSGM16a]. The second group of OMT problems was generated by the PYLMT [pyl] framework and it is based on (Machine) Learning Modulo Theories [TSP17]. Three out of four of the OMT($\mathcal{PB} \cup \mathcal{T}$)/MAXSMT benchmark-sets considered here encode a problem that requires the capability of dealing with the Theory of $\mathcal{LIRA}$. The fourth benchmark-set, used in §6.2.2, is the only one that can be restricted to the Theory of $\mathcal{LIA}$. We evaluated the performance of both OPTIMATHSAT and Z3 over the OMT($\mathcal{LIRA} \cup \mathcal{T}$) formulas. When dealing with the OMT($\mathcal{LIA} \cup \mathcal{T}$) benchmark-set, we included also BCLT in our experiment.

We used version 1.3.11 of OPTIMATHSAT for the experiment in §6.2.4 and version 1.4.1 for all others. When dealing with OMT formulas suitable for the OMT-based techniques, we experimented with three different approaches: "LA", the default OMT encoding described in §2.3.3, "SEQ", the *bidirectional sequential counter* encoding described in §4.2.1, and "CAR", the *cardinality sorting network* encoding described in §4.2.1. When dealing with OMT problems suitable for MAXSAT-based approaches, two additional configurations have been tested. The first is the MAXRES engine of [NB14, BP14] implemented on top of OPTIMATHSAT as described in §4.2.2. The second, applicable only on pure MAXSMT formulas, is the lemma-lifting approach of [CGSS13a], that we have also implemented on top of OPTIMATHSAT using

MAXINO [ADR15] as external MAXSAT engine.

For what concerns Z3, we used version $4.4.2$ of this software and tested three different configurations. This includes the default OMT-based approach of Z3, based on linear arithmetic, the MAXRES engine [NB14, BP14], which is a MAXSAT-based approach, and WMAX[NO06]. As mentioned in [ST17], the technique used by Z3 for dealing with OMT($\mathcal{PB} \cup \mathcal{T}$)/MAXSMT problems largely depends on the encoding of the input formula. Therefore, the subset of configurations used for Z3 varies with each experiment.

6.2.1 CGMs with lexicographic OMT

Experiment Setup. The benchmark-set consists of 18996 formulas automatically-generated with the CGM-TOOL in [NSGM16a]. Each formula is a lexicographic OMT($\mathcal{LIRA} \cup \mathcal{T}$) instance $\langle \varphi, \mathsf{obj}_1, ..., \mathsf{obj}_N \rangle_{\mathcal{L}}$ (§4.6.3), with N ranging from 1 to 3, that solves the problem of computing the lexicographically-optimum realization of a Constrained Goal Model [NSGM16a]. Due to the large number of formulas, in this experiment each solver configuration was allowed to execute for up to $100s$., and then forced to terminate.

> **Note.** In the original experimental evaluation, reported in [ST17], we have used version 4.4.2 of Z3. However, there was a bug in the lexicographic engine of this version of the tool that prevented us from testing Z3 on this set of benchmarks using WMAX. Since this issue has been recently solved, we have repeated the experiment with version 4.8.1 of Z3 using both optimization engines. The results are reported in Table 6.2 for a comparison.

Experiment Results. A summary of the performance results of both OPTIMATHSAT and Z3 on this benchmark-set is shown in table 6.2.

As a first observation, we note that as far as OPTIMATHSAT is concerned with extending the input formula with either sorting networks increases the number of benchmarks solved within the timeout. When the cardinality network encoding is used, this results in an improvement of both the number of formulas being solved as well as of the solving time as a whole. In contrast, using the sequential counter network results in a significant performance hit on a number of benchmarks, as it is witnessed by the data in table 6.2 and the top-left plot in Figure 6.2. This does not only affect unsatisfiable benchmarks, for which the use of sorting networks appears to be not beneficial in general, but also satisfiable ones.

The performance gap among the two sorting network encodings is explained by the lower space complexity of the cardinality network encoding used in [ANORC11, ANOR13]. This gap can be reduced by limiting the size of the sorting network circuit that is generated at the runtime,

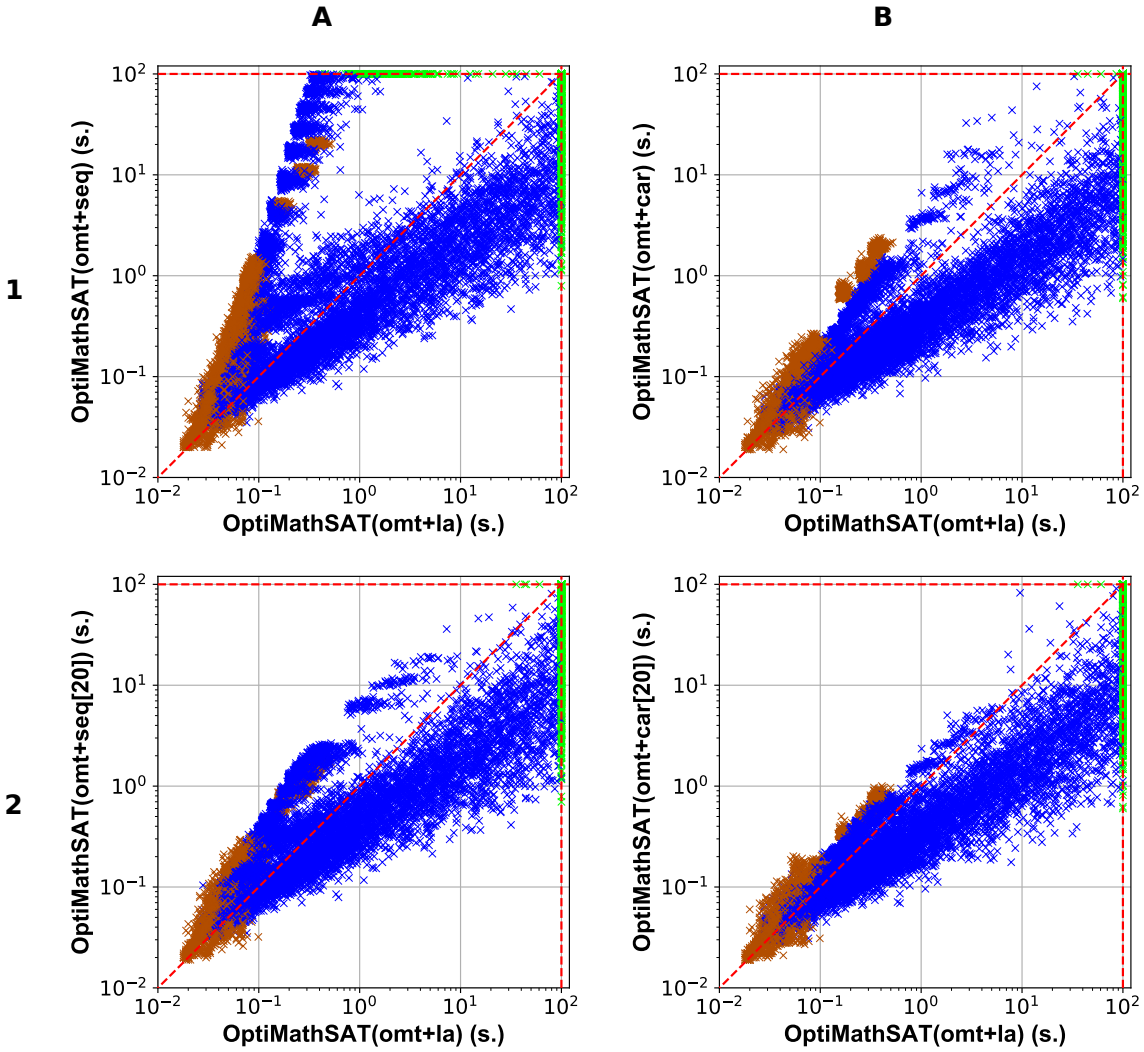


Figure 6.2: Pairwise comparisons on lexicographic OMT formulas of [NSGM16b, NSGM16a] without a limit on the circuit size (top) and with a limit of 20 components (bottom). (Brown points denote unsatisfiable benchmarks, blue denote satisfiable ones and green ones represent timeouts.).

tool, configuration & encoding	inst.	term.	timeout	time (s.)
OPTIMATHSAT(OMT+LA)	18996	16316	2680	48832
OPTIMATHSAT(OMT+SEQ)	18996	16929	2067	90080
OPTIMATHSAT(OMT+CAR)	18996	**17191**	1805	39215
OPTIMATHSAT(MAXRES+LA)	18996	17933	1063	24369
OPTIMATHSAT(MAXRES+SEQ)	18996	18180	816	49088
OPTIMATHSAT(MAXRES+CAR)	18996	**18197**	799	26489
Z3(MAXRES) [V. 4.4.2]	18996	**18996**	0	**1640**
Z3(MAXRES) [V. 4.8.5]	18996	**18996**	0	1929
Z3(WMAX) [V. 4.8.5]	18996	18421	575	29401

Table 6.2: Comparison between various OPTIMATHSAT configurations and Z3 on the lexicographic OMT formulas of [NSGM16b, NSGM16a].

(OMT-based)		split sequential-counter enc.			split cardinality-network enc.		
vars	inst.	term.	timeout	time (s.)	term.	timeout	time (s.)
∞	18996	16929	2067	90080	**17191**	1805	39215
10	18996	17033	1963	39035	17058	1938	36636
15	18996	17061	1935	39264	17133	1863	37246
20	18996	**17152**	1844	43730	17190	1806	39492

Table 6.3: Effect of splitting the PB sums into chunks of maximum variable number (no split, 10, 15, 20 variables) with the sequential-counter encoding and the cardinality-network encoding.

so that the OMT solver keeps track of a fewer number of (smaller) clauses and variables. To achieve this goal, it suffices to slice each Pseudo-Boolean sum into a number of smaller sized chunks, and then generate a separate sorting circuit for each splice. Table 6.3 shows the result of applying this technique using chunks of size up to 10, 15 and 20 components respectively. The latter configuration, limiting the circuit to up to 20 components, it is also show-cased in the bottom-left plot of Figure 6.2. The experimental data suggest that the sequential counter encoding can benefit from this simple heuristic, both in terms of number of benchmarks being solved and solving time. Among the two sorting networks, the leadership is maintained by the cardinality network encoding, which however does not seem to be particularly affected by the splitting strategy.

Overall, we observe that when using either optimization engine Z3 outperforms OPTIMATHSAT and, when using the MAXRES engine, it solves all problems in the benchmark-set.

tool, configuration & encoding	inst.	term.	timeout	time (s.)
BCLT()	2499	1997	502	7194
OPTIMATHSAT(OMT+LA)	2499	1794	705	11178
OPTIMATHSAT(OMT+SEQ)	2499	**2451**	48	18033
OPTIMATHSAT(OMT+CAR)	2499	2186	313	10633
OPTIMATHSAT(MAXRES+LA)	2499	**2499**	0	**128**
OPTIMATHSAT(MAXRES+SEQ)	2499	2499	0	1638
OPTIMATHSAT(MAXRES+CAR)	2499	2499	0	257
OPTIMATHSAT(LEMMA-LIFTING+MAXINO)	2499	2497	2	343
Z3(MAXRES)	2499	**2499**	0	**119**
Z3(WMAX)	2499	1799	733	10549

Table 6.4: Results of various solvers, configurations and encodings on un-weighted MAXSMT formulas derived from the Constrained Goal Model instances of [NSGM16b, NSGM16a].

6.2.2 CGMs with Un-weighted MAXSMT

Experiment Setup. The benchmark-set used in this experimental evaluation is a variant of the one used in §6.2.1, in which the weight of all soft clauses has been made equal to 1 and the formulas containing more than one objective function were discarded. This yielded 2499 OMT($\mathcal{LIA} \cup \mathcal{T}$) benchmarks in total, each of which is an instance of an un-weighted MAXSMT problem. The search timeout for this experiment was set to $100s$..

Experiment Results. The results of this experiment are shown in table 6.4.

We observe that, similarly to the previous experiment in §6.2.1, OPTIMATHSAT benefits from extending the input formula with either sorting networks when using OMT-based techniques. In fact, the data show a drastic increase in the number of formulas solved within the timeout, which is particularly accentuated when the sequential counter encoding is used.

Using OPTIMATHSAT with any of the available MAXSAT-based techniques significantly outperforms any variant based on the standard OMT search. In particular, OPTIMATHSAT solves all of the benchmark-set instances when using the MAXRES engine and all-but-two formulas when MAXINO is used.

As far as Z3 is concerned, we observe that it obtains the best score when using the MAXRES engine, although the differences with respect to OPTIMATHSAT(MAXRES +LA) is negligible. When using the WMAXengine, its performances are comparable to that of OPTIMATHSAT with a standard OMT-based search.

tool, configuration & encoding	inst.	term.	timeout	time (s.)
OPTIMATHSAT(OMT+LA)	2399	2340	59	20841
OPTIMATHSAT(OMT+SEQ)	2399	2394	5	9511
OPTIMATHSAT(OMT+CAR)	2399	**2395**	4	8275
Z3(OMT +LA)	2399	2390	9	8076

Table 6.5: Results of various solvers with OMT-based techniques on MAXMIN formulas derived from the Constrained Goal Model instances of [NSGM16b, NSGM16a] .

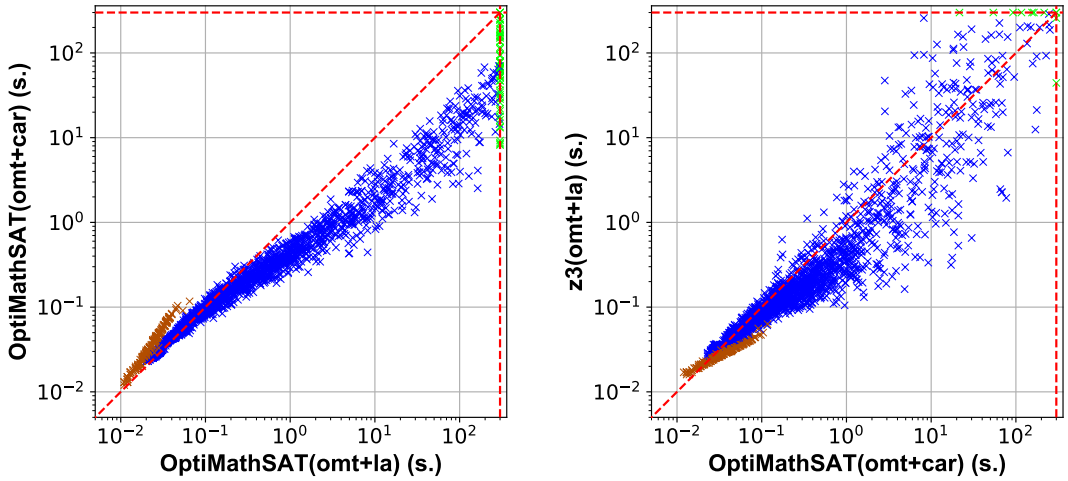


Figure 6.3: Pairwise comparison on MAXMIN formulas from [NSGM16b, NSGM16a]. (Brown points denote unsatisfiable benchmarks, blue denote satisfiable ones and green ones represent timeouts.).

6.2.3 CGMs with MAXMIN OMT

Experiment Setup. In this variant of the experiment in §6.2.1, we first selected only those formulas containing at least three objectives obj_i and then normalized the range of each obj_i to be included in the range $[0, 1]$ (i.e., obj_i was divided by $\sum_j w_{ij}$). Then, we searched for the optimal model $\mathcal{M}$ that maximizes the minimum gain for each normalized objective obj_i. This corresponds to solving a multi-objective OMT problem subject to the MAXMIN combination described in §4.6.1. We note that in this evaluation we considered only those formulas that originally contained at least three objectives, so that the resulting benchmark-set was comprised by 2399 OMT($\mathcal{LIRA} \cup \mathcal{T}$) problems. The search timeout was increased to $300s$., to account for the more complex optimization goal.

tool, configuration & encoding	inst.	term.	timeout	time (s.)
OPTIMATHSAT(OMT+LA)	500	421	79	2607
OPTIMATHSAT(OMT+SEQ)	500	441	59	6381
OPTIMATHSAT(OMT+CAR)	500	**442**	58	6189
Z3(OMT+LA)	500	406	94	2120

Table 6.6: Comparison among various OPTIMATHSAT configurations and Z3 on a set of benchmarks based on the Structured Learning Modulo Theories domain [TSP17].

Experiment Results. The results are reported in table 6.5, whilst figure 6.3 shows the pairwise comparisons of OPTIMATHSAT(OMT+LA) with respect to OPTIMATHSAT(OMT+CAR) (left) and Z3(OMT) with respect to OPTIMATHSAT(OMT+CAR) (right). Looking at the data, we observe that using either sorting networks encoding significantly improves the performance. In addition, the right scatter-plot in Figure 6.3 suggest that OPTIMATHSAT performs equivalently or slightly better than Z3 with the aid of the cardinality network encoding, when both use OMT-based techniques.

6.2.4 LMT with OMT($\mathcal{PB} \cup \mathcal{T}$)

Experiment Setup. The benchmark-set used in this experiment is comprised by 500 formulas generated by PYLMT [pyl]. As described in §7.2.1, PYLMT is a tool for Structured Learning Modulo Theories [TSP17] that uses OPTIMATHSAT as back-end engine for doing inference in the context of machine learning in hybrid domains. Each problem in this benchmark-set is an OMT($\mathcal{PB} \cup \mathcal{T}$) instance wherein obj is a $\mathcal{LRA}$ term defined as

$$\mathsf{obj} \stackrel{\text{def}}{=} \sum_j w_j \cdot B_j + cover - \sum_k w_k \cdot C_k - |K - cover|, \tag{6.1}$$

$$\text{such that } cover \stackrel{\text{def}}{=} \sum_i w_i \cdot A_i \tag{6.2}$$

A_i, B_j, C_k being Boolean atoms, w_j, w_k, w_i being rational constants. Each OMT solver configuration was allowed to run for up to $600s$. to solve each instance in this experiment.

Experiment Results. Table 6.6 and Figure 6.4 show the performance results for OPTIMATHSAT and Z3 on this experiment.

Looking at the experimental data, we observe that the use of sorting networks provides a limited improvement to the performance of OPTIMATHSAT as a whole. As mentioned in [ST17], we conjecture that this is due to the high diversity among the values of the weights w_i, w_j, w_k

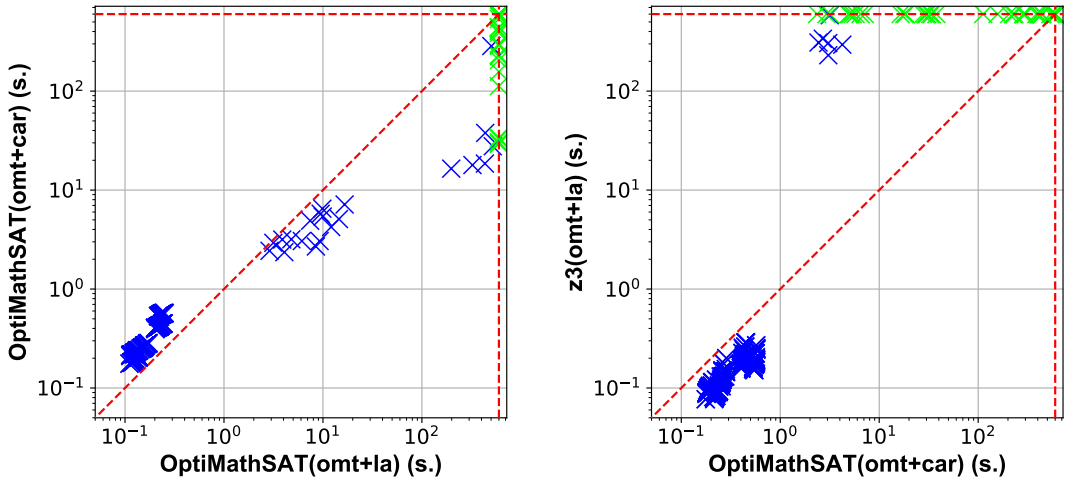


Figure 6.4: Pairwise comparisons on OMT($\mathcal{PB} \cup \mathcal{T}$) formulas from [TSP17]. (Blue points denote satisfiable benchmarks, green denotes a timeout.)

appearing in the formulas. In fact, as described in §4.2.1, the sorting network enhancement is most effective when many Pseudo Boolean terms share the same weight.

6.3 OMT ($\mathcal{BV} \cup \mathcal{T}$) Evaluation

In this experimental evaluation, we assessed the performance of OPTIMATHSAT over a subset of the OMT($\mathcal{BV} \cup \mathcal{T}$) formulas used by Nadel et al. in [NR16].

Experiment Setup. In this experiment, we used the default *testing workbench* described at the beginning of Chapter §6. Similarly to [NR16], we set a timeout of $1800s$. on each formula. Differently than in [NR16], we have included only the $\mathcal{BV}$-encoded version of the formulas used in the original experiment. This is because, in their paper, Nadel et al. have shown that OMT tools tend to perform poorly on this particular benchmark-set when the $\mathcal{LIA}$-based encoding is used. Moreover, we have restricted our evaluation on the subset of 254 *crafted* benchmarks used in [NR16], as the *industrial* ones —together with the original tools used in [NR16]— have not been made publicly available[31]. We recall here that each instance in this benchmark-set encodes an OMT($\mathcal{BV} \cup \mathcal{T}$) formula wherein the cost function is an *unsigned* $\mathcal{BV}$.

The following OPTIMATHSAT (v. 1.5.1) configurations were included in our experimental evaluation (For more details on each $\mathcal{BV}$-optimization technique, see Section §4.3):

- OPTIMATHSAT(LIN): OMT-based linear search with $\mathcal{BV}$ objective.

[31] Source: private communication with one of the authors.

- OPTIMATHSAT(BIN): OMT-based binary search with $\mathcal{BV}$ objective.
- OPTIMATHSAT(OBV-BS): the extended version of the OBV-BS algorithm in [NR16], implemented on top of OPTIMATHSAT.

For each configuration, we separately tested the effect of enabling the *branching preference* (BP) and *polarity initialization* (PI) enhancements described in §4.3. We recall here that, as described in §4.3.2, enabling both of these enhancements at the same time forces OPTIMATHSAT—regardless of the selected $\mathcal{BV}$ optimization engine— to behave as in the OBV-WA algorithm. This means that the OMT solver is forced to approach the optimal solution starting from the unsatisfiable region, and advance step-by-step towards the optimal solution without ever touching any suboptimal satisfiable solution.

In addition to OPTIMATHSAT, we included in our experiment Z3 (v. 4.6.0) and ran it with its default configuration. Unfortunately, we were unable to include in our experimentation the original OBV-WA and OBV-BS binaries used in [NR16], as neither of these tools was made publicly available by the authors[31].

We have independently verified the correctness of the optimal solution found by each OMT configuration with a third-party SMT solver (MATHSAT5). To perform this cross-check as efficiently as possible, we have enabled *model generation* on every configuration so that the optimum model could be extracted and verified.

Experiment Results. The results of this experimental evaluation are summarized in table 6.7, that includes a log-scale cactus plot for a visual comparison among the different configurations.

We start by looking at the results for the LIN and BIN configurations, that are very similar to each other. In both cases, it can be seen that approaching the optimal solution starting from the unsatisfiable region yields better performance than proceeding from the satisfiable region. In fact, enabling both *branching preference* and *polarity initialization* nearly doubles the number of formulas being solved for both the LIN and BIN configurations. On this regard, we note that when the BP+PI combination is used then we get pretty uniform performance results with each configuration being tested (that is, LIN, BIN and OBV-BS). This is witnessed by the data in table 6.7, by the overlapping BP+PI lines in the cactus plot of Figure 6.5, and also by the sixth scatter plot in Figure 6.5. This supports our previous observation that enabling both *branching preference* and *polarity initialization* forces OPTIMATHSAT to behave as in OBV-WA regardless of the $\mathcal{BV}$ engine being selected. As observed by Nadel et al. in [NR16], the performance improvement obtained with OBV-WA is likely due to the fact that on this particular benchmark-set the optimal solution of the $\mathcal{BV}$ objective lies very close to the maximum representable value

tool, configuration & encoding	inst.	term.	timeout	time (s.)
OPTIMATHSAT(LIN)	254	53	201	6085
OPTIMATHSAT(LIN+BP)	254	57	197	11326
OPTIMATHSAT(LIN+PI)	254	55	199	9456
OPTIMATHSAT(LIN+BP+PI) [i.e. OBV-WA]	254	108	146	24917
OPTIMATHSAT(BIN)	254	62	192	12113
OPTIMATHSAT(BIN+BP)	254	55	199	8896
OPTIMATHSAT(BIN+PI)	254	62	192	11840
OPTIMATHSAT(BIN+BP+PI) [i.e. OBV-WA]	254	108	146	24905
OPTIMATHSAT(OBV-BS)	254	80	174	12548
OPTIMATHSAT(OBV-BS+BP)	254	69	185	11668
OPTIMATHSAT(OBV-BS+PI)	254	**148**	106	**17860**
OPTIMATHSAT(OBV-BS+BP+PI) [i.e. OBV-WA]	254	108	146	24892
Z3	254	129	125	20279
VIRTUAL BEST	254	**148**	106	**17846**

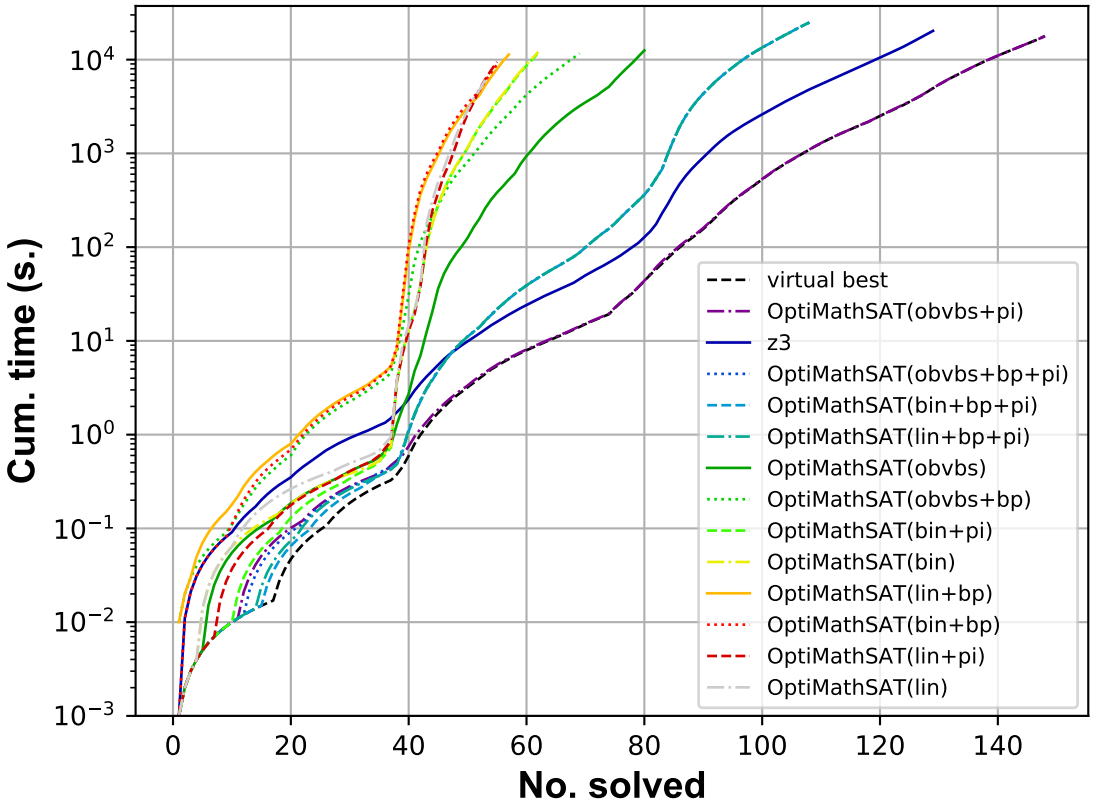


Table 6.7: Comparison among various OPTIMATHSAT configurations and Z3 on a subset of 254 OMT($\mathcal{BV} \cup \mathcal{T}$) formulas taken from [NR16].

with the $\mathcal{BV}$ variable itself. This makes it more convenient to explore the search space from within the unsatisfiable *more-than-optimal* region.

We can also see that there is relatively small performance improvement when *branching preference* and *polarity initialization* are enabled independently from one another over the LIN configuration. This is to be expected because the set of pseudo-random and heuristic techniques employed by the underlying CDCL solver result in a relatively random exploration of the search space that can hit a large number of suboptimal solutions before reaching the optimal one. The situation is slightly different for what concerns the BIN configuration, that uses a sequence of *pivoting* steps to perform greater forward leaps towards the optimal solution. In this case, enabling only *branching preference* is detrimental with respect to to the overall performance because it delays the evaluation of each *pivoting* cut after the CDCL engine already decided to assign some value to each bit of the $\mathcal{BV}$ objective. In the case the former set of decisions conflict with the *pivoting* cut, the CDCL engine is forced to resolve each of these conflicting decisions before that the rest of the input formula —which is where the search should focus— can be taken into consideration.

Overall, the best performance is achieved by OPTIMATHSAT(OBV-BS+PI), that solves 148 formulas out of 254 within the timeout. As shown in table 6.7, we observe that OPTIMATHSAT(OBV-BS+PI) solves as many formulas as the computed VIRTUAL BEST configuration with only a small disadvantage in terms of running time (14 seconds). The first four scatter plots in Figure 6.5 provide a more detailed view on the performance of OPTIMATHSAT(OBV-BS+PI) by comparing it with the best results obtained with the other configurations evaluated in this experiment. In particular, we observe more than an order of magnitude improvement with respect to the best LIN and BIN configurations, and a more contained advantage with respect to the OBV-WA (a.k.a. BP+PI) optimization search. Moreover, the best configuration of OPTIMATHSAT performs significantly better than Z3 with its default configuration. The fifth scatter plot in Figure 6.5, instead, provides a clear view of the overwhelming contribution of enabling *polarity initialization* to the overall performance of the OBV-BS engine. This result is not surprising. As a matter of fact, since the optimal solution on this set of benchmarks is very close to the maximum representable value with the $\mathcal{BV}$ objective (see [NR16]), the initial branching value that is being decided by the CDCL engine with the *polarity initialization* enhancement turns out to be a good choice more often than not. By looking at the table in Figure 6.7, we note also that OBV-BS performs better without the *branching preference* enhancement. Similarly to the case of the BIN configuration, executing without BP allows the CDCL solver to let the structure of the underlying input formula to lead the decision process over the bits of the $\mathcal{BV}$ objective, as opposed to forcing a pseudo-random set of decisions that do not focus the optimization search in any specific direction.

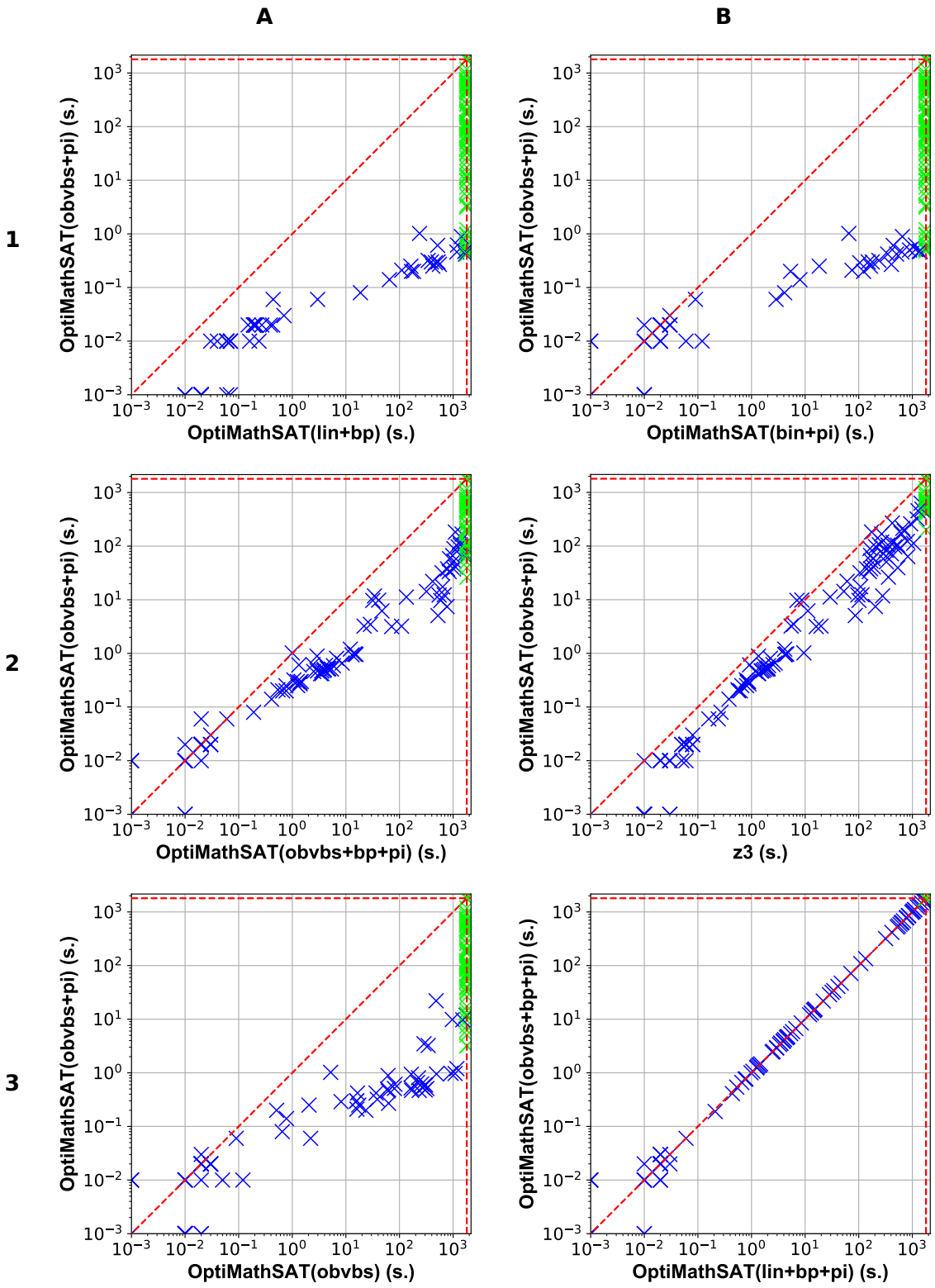


Figure 6.5: Pairwise comparisons on OMT($\mathcal{BV} \cup \mathcal{T}$) formulas of [NR16] among various OPTIMATHSAT configurations and Z3. (Blue points denote satisfiable benchmarks, green denotes a timeout.)

Comparison with [NR16]. On the whole, the results of this experimental evaluation seem to be compatible with those of [NR16], and lead to similar conclusions as in the original work. In addition, our experiment provides a more detailed overview of the performance impact of two optimization search enhancements, namely *branching preference* and *polarity initialization*, proposed in [NR16] and also implemented in OPTIMATHSAT as described in §4.3.

However, we observe that our implementation of the OBV-WA and OBV-BS algorithms on top of OPTIMATHSAT are unable to attain as good performance results as those shown in [NR16]. For instance, the OBV-BS engine with *polarity initialization* used in [NR16] was able to solve within the timeout all 254 formulas being considered, whereas in our experiment it successfully solves only 148 of them. At the time being the reason for this discrepancy is still unknown and it remains difficult to determine in practice, because the tool of [NR16] is not available. Note that we are using a different hardware and experimental setup than in the original experiment. Moreover, we implemented these engines on top of a different solver and we introduced some minor changes to deal with a more general definition of the optimization problem. Unfortunately, we are unable to compare with the original OBV-WA and OBV-BS implementations as these were not made available by the authors.

6.4 OMT ($\mathcal{FP} \cup \mathcal{T}$) Evaluation

We assess the performance of OPTIMATHSAT on a set of OMT($\mathcal{FP}$) formulas that were generated using the SMT($\mathcal{FP}$) benchmark-set of [smt].

Experiment Setup. Due to resource constraints, this experiment was not performed on the default *testing workbench*. Instead, we used an $i7-6500U\ 2.50GHz$ Intel Quad-Core machine with $16GB$ of ram and running *Ubuntu Linux* 17.10. Moreover, for each formula being tested we used a timeout of 600 seconds.

The OMT($\mathcal{FP}$) instances used in this experiment were automatically generated starting from the satisfiable formulas included in the SMT($\mathcal{FP}$) benchmark-set of [smt]. We did not consider any of the unsatisfiable instances that are present in the remote repository. For each of the original SMT($\mathcal{FP}$) formulas we applied the following transformations. First, we either relaxed or removed some of the constraints in the original problem, to broaden the set of feasible solutions. This step is necessary because the majority of the original SMT($\mathcal{FP}$) formulas admits only one solution. However, this is not necessarily the ideal situation when comparing different optimization approaches. Second, for each $\mathcal{FP}$ variable v appearing inside a SMT($\mathcal{FP}$) problem we generated a pair of OMT($\mathcal{FP}$) instances, one for the minimization and another for the maximization of v. At the end of this step, we obtained 39536 OMT($\mathcal{FP}$)

formulas. Third, we randomly selected up to 300 OMT($\mathcal{FP}$) instances from each of the five groups of problems in the OMT($\mathcal{FP}$) benchmark-set. This filtering step yielded a total of 1120 SMT-LIBV2 formulas.

Using OPTIMATHSAT (v. 1.6.2), we consider two OMT-based baseline configurations, OPTIMATHSAT(OMT+LIN) and OPTIMATHSAT(OMT+BIN), that run the linear- and the binary-search respectively. These configurations have been tested using both the *eager* and the *lazy* $\mathcal{FP}$ approaches. The third baseline approach, named OPTIMATHSAT(EAGER+OBV-BS), is based on a reduction of the OMT($\mathcal{FP}$) problem to OMT($\mathcal{BV}$) and it uses OPTIMATHSAT's implementation of the OBV-BS engine[32] presented by Nadel et al. in [NR16]. For this test, we have generated an OMT($\mathcal{BV}$) benchmark-set using a $\mathcal{BV}$ encoding that mimics the essential aspects of the OFP-BS algorithm described Section §4.4.2.

We compared these baseline approaches with a configuration using the OFP-BS algorithm and the *eager* $\mathcal{FP}$ approach, namely OPTIMATHSAT(EAGER+OFP-BS).

We have separately tested the effect of enabling the *branching preference* (BP), the *polarity initialization* (PI) and the *safe bits restriction* (SO) enhancements described in Section §4.4, whenever these options were supported by the given configuration.

Last, in order to assess the significance of the optimization problems used in this experiment, we have collected the run-time statistics of OPTIMATHSAT on the SMT formulas obtained by stripping the objective function from each OMT instance. We named this configuration OPTIMATHSAT(EAGER+SMT).

We have not included other tools in our experiment because we are not aware of any other OMT($\mathcal{FP}$) solver.

We have independently verified the correctness of the optimal solution found by each configuration with a third party SMT solver (MATHSAT5). To perform this cross-check as efficiently as possible, we have enabled *model generation* on every configuration so that the optimum model could be extracted and verified.

Experiment Results. The results of this experiment are listed in Table 6.8, that includes a log-scale cactus plot for a visual comparison among the different configurations. In addition, Figures 6.6, 6.7 and 6.8 show a selection of relevant pairwise comparisons among various OPTIMATHSAT configurations. Figure 6.6 focuses on variants of the OMT-based linear-search approach, Figure 6.7 depicts variants of the OMT-based binary-search approach, whereas Figure 6.8 focuses on the OFP-BS engine.

For what concerns OMT-based *linear-search* optimization, we observe that OPTIMATHSAT performs the best when no enhancement is enabled. In particular, the empirical evidence

[32]The binaries of the original OMT($\mathcal{BV}$) tools presented in [NR16] are not publicly available.

configuration & encoding	inst.	term.	t.o.	u	bt	st	time (s.)
EAGER+OMT+LIN	1120	1003	117	0	5	73	76375
EAGER+OMT+LIN+PI	1120	1003	117	0	5	71	76785
EAGER+OMT+LIN+BP	1120	956	164	0	6	105	77480
EAGER+OMT+LIN+BP+PI	1120	873	247	0	77	217	54859
EAGER+OMT+BIN	1120	1014	106	0	11	281	67834
EAGER+OMT+BIN+PI	1120	970	150	0	8	285	69765
EAGER+OMT+BIN+BP	1120	1016	104	0	14	205	68255
EAGER+OMT+BIN+BP+PI	1120	991	129	0	65	**321**	56941
LAZY+OMT+LIN	1120	868	252	0	93	203	29832
LAZY+OMT+BIN	1120	900	220	0	90	243	33260
EAGER+OBVBS [REDUCTION]	1120	1013	107	0	14	141	65954
EAGER+OFPBS	1120	1017	103	0	9	171	70732
EAGER+OFPBS+PI	1120	**1019**	101	0	34	280	64896
EAGER+OFPBS+PI+SO	1120	1018	102	0	7	179	71430
EAGER+OFPBS+BP	1120	975	145	0	2	145	65543
EAGER+OFPBS+BP+SO	1120	1000	120	0	3	124	68390
EAGER+OFPBS+BP+PI	1120	1001	119	0	77	273	60365
EAGER+OFPBS+BP+PI+SO	1120	1006	114	**19**	32	245	59463
VIRTUAL BEST	1120	**1074**	46	-	559	1074	27788
EAGER+SMT [NO OPTIMIZATION]	1120	1048	72	-	-	-	9259

Table 6.8: Comparison among various OPTIMATHSAT configurations on a subset of 1120 OMT($\mathcal{FP}$) formulas generated from the SMT($\mathcal{FP}$) formulas of [smt]. The columns list the total number of instances (inst.), the number of instances solved (term.), the number of timeouts (t.o.), the number of instances uniquely solved by the given configuration (u), the number of instances solved faster than any other configuration (bt), the total number of instances solved in the shortest amount of time (st) and the total solving time for all solved instances (time).

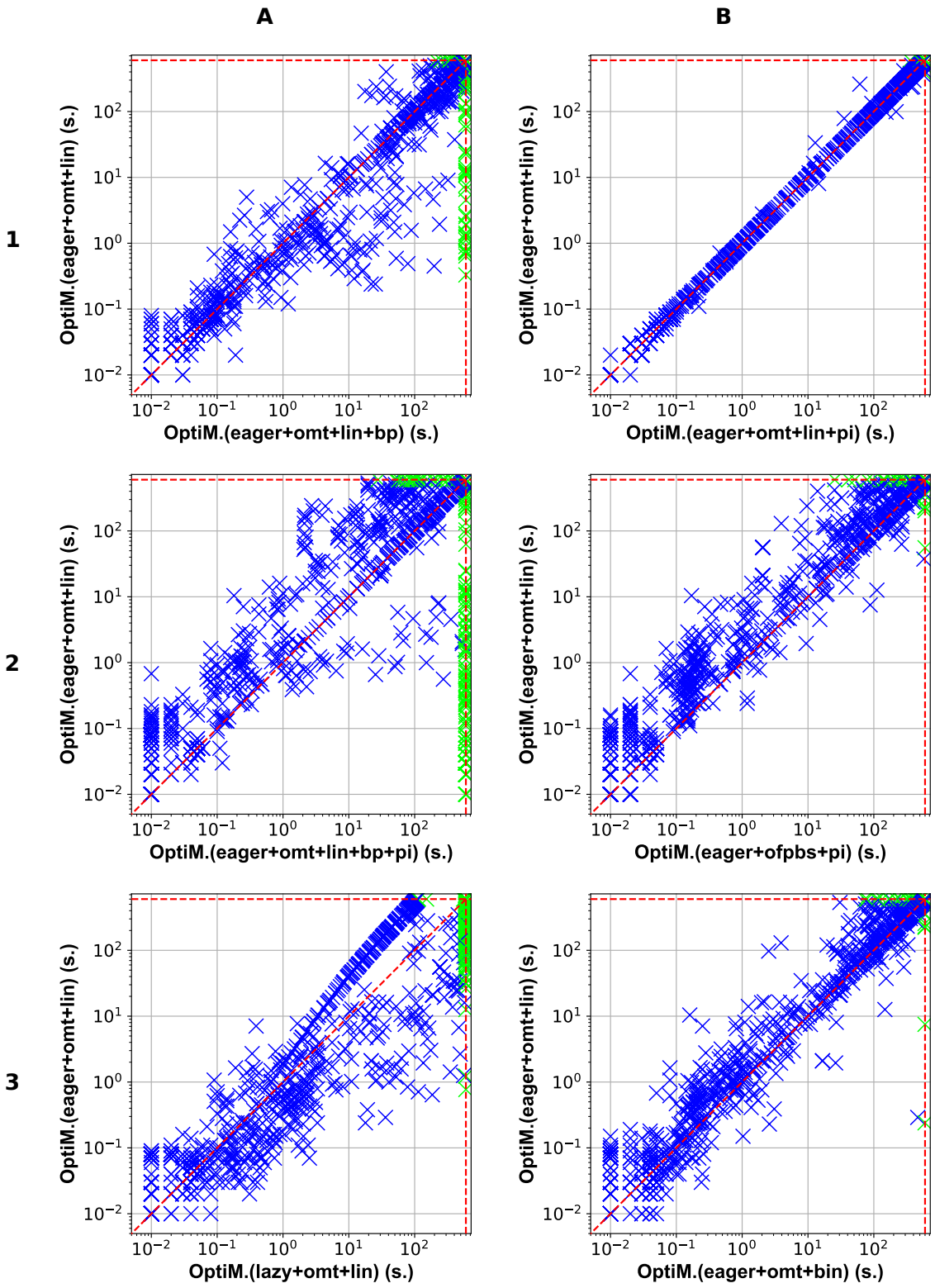


Figure 6.6: Pairwise comparisons on OMT($\mathcal{FP}$) formulas using OMT-based linear-search and other configurations. (Blue points denote satisfiable benchmarks, green denotes a timeout.)

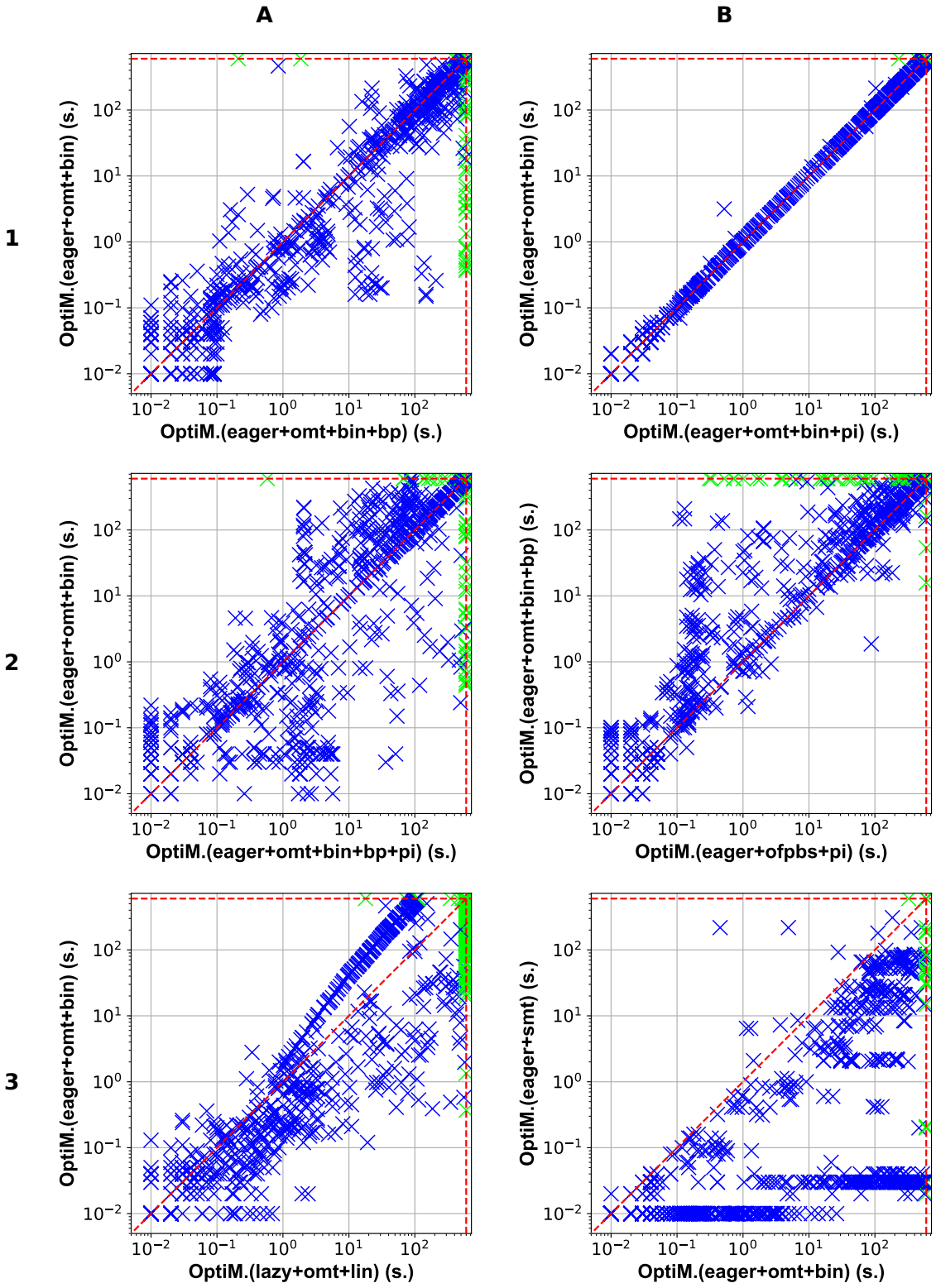


Figure 6.7: Pairwise comparisons on OMT($\mathcal{FP}$) formulas using OMT-based binary-search and other configurations. (Blue points denote satisfiable benchmarks, green denotes a timeout.)

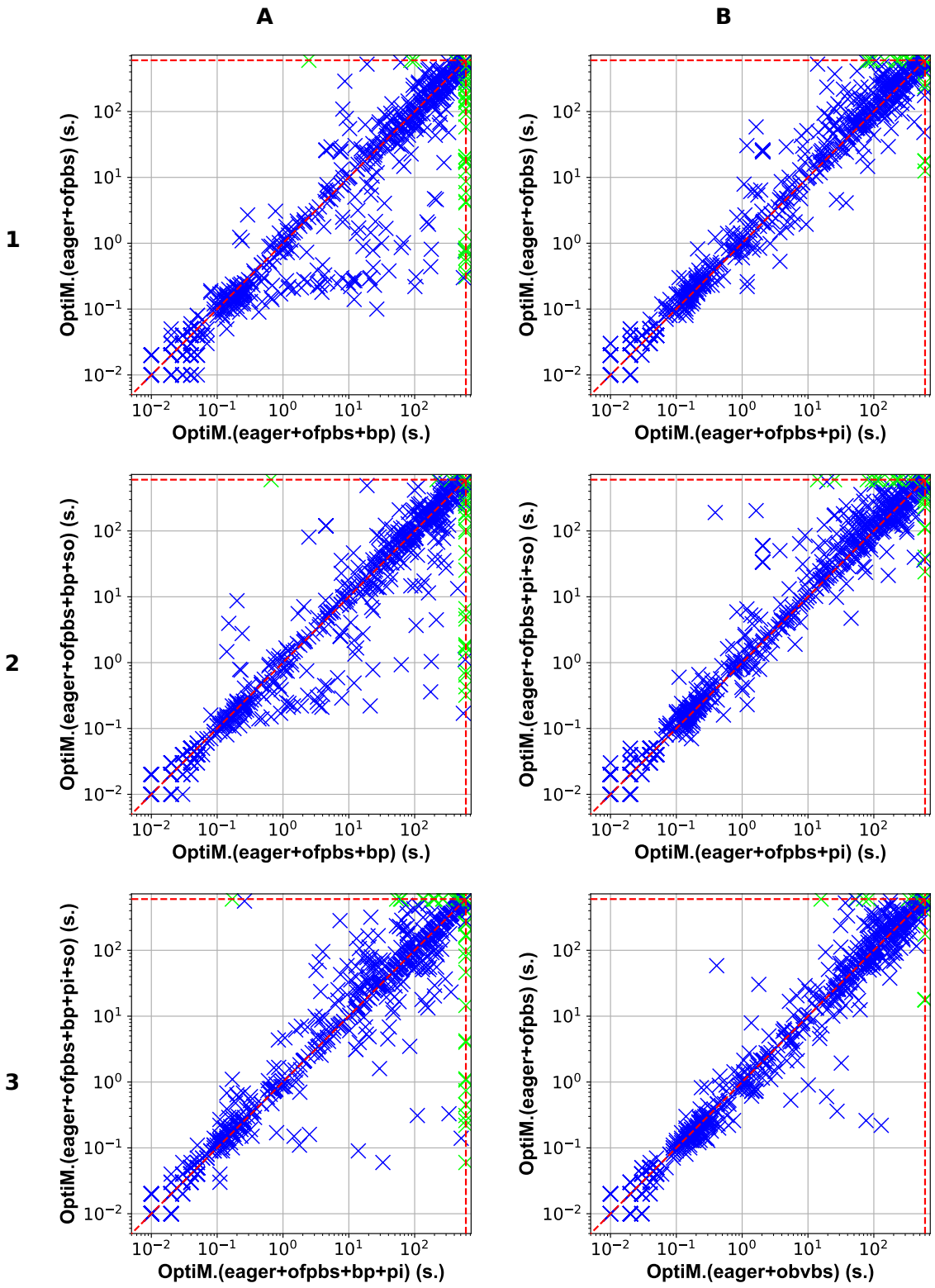


Figure 6.8: Pairwise comparisons on OMT($\mathcal{FP}$) formulas using the OFP-BS engine and other configurations. (Blue points denote satisfiable benchmarks, green denotes a timeout.)

suggests that enabling *branching preference* significantly increases the number of timeouts, generally deteriorating the performance (plot $1A$ in Fig. 6.6). Enabling only *polarity initialization* does not result in an appreciable change on the running time of the solver (plot $1B$ in Fig. 6.6). In contrast, enabling both enhancements at the same time has a small chance to result in a small improvement of the search time (plot $2A$ in Fig. 6.6), but it generally worsens the performance and results in a drastic increase in the number of timeouts (Table 6.8). We justify these results as follows. First, when only *polarity initialization* is used, the phase-saving value that is being set by OPTIMATHSAT does not really matter because the optimization search is dominated by the structure of the formula itself rather than by the bits $[\mathsf{obj}[0], \ldots, \mathsf{obj}[n-1]]$ of the $\mathcal{FP}$ objective. Second, when *polarity initialization* is used on top of *branching preference*, there is an even more drastic decrease in performance due to the fact that the initial phase-saving value that is statically assigned by the OMT solver to the bits of the $\mathcal{FP}$ objective cannot be expected to be "good enough" for any situation. In fact, as illustrated in Example 4.4.1, the initial phase-saving can be misleading and force the OMT solver —when running in *linear-search*— to explore an exponential number of intermediate satisfiable solutions.

In the case of the OMT-based *binary-search* optimization approach, we observe that it solves more formulas than linear-search and it generally appears to be faster (plot $3B$ in Fig. 6.6). Overall, *polarity initialization* does not seem to be beneficial, whereas enabling *branching preference* increases the number of formulas solved within the timeout. This behavior is different from the linear-search approach, and we conjecture that it is due to the fact that, with the OMT-based binary-search approach, branching over the bits of the objective function can reveal in advance any (partial) assignment to the bits of the objective function that it is inconsistent wrt. the pivoting cuts learned by the optimization engine.

Using the *lazy* $\mathcal{FP}$ engine results in fewer formulas being solved within the timeout, although a significant number of these benchmarks is solved faster than with any other configuration (over 90 instances, for both configurations).

The OPTIMATHSAT(EAGER+OBV-BS) configuration is able to solve 1013 formulas within the timeout, showing that OMT($\mathcal{FP}$) can be reduced to OMT($\mathcal{BV}$) effectively, and that – on the given benchmark-set– the performance of this approach are comparable with the best OMT($\mathcal{FP}$) configurations being tested.

Overall, the best performance is obtained by using the OFP-BS engine, with up to 1019 benchmark-set instances being solved in correspondence to the OPTIMATHSAT(EAGER+OFP-BS+PI) configuration. In plot $2B$ of Figures 6.6 and 6.7, we show the pairwise comparison of the best OFP-BS configuration with the best OMT-based run. Similarly to the case of OMT-based optimization with linear-search, we observe that enabling *branching preference* generally makes the performance worse (plot $1A$ in Fig. 6.8). Instead, when *polarity initialization* is used

we observe a general performance improvement that does not only result in an increase in the number of formulas being solved within the timeout, but also a noticeable reduction of the solving time as a whole. This is in contrast with the case of OMT-based optimization, and it can be explained by the fact that OFP-BS uses an internal heuristic function to dynamically determine and update the most appropriate phase-saving value for the bits of the objective function (i.e. the *dynamic attractor* described in §4.4). An equally important role is played by the *safe bits restriction*, that limits the effects of *branching preference* and *polarity initialization* to only certain bits of the *dynamic attractor*. As illustrated by the plots in the second and third rows of Figure 6.8 and by the data in Table 6.8, this feature is particularly effective when used in combination with *branching preference*.

The results of OPTIMATHSAT over the SMT-only version of the benchmark-set are reported in Table 6.8 and in the scatter-plot $3B$ in Fig. 6.7, and show that for a large number of instances the OMT problem is considerably harder than its SMT-only version There are a few exceptions to this rule, that we ascribe to the fact that the removal of the objective function alters the internal stack of formulas, and this can have unpredictable consequences on the behavior of various internal heuristics that depend on it. A solution can be found in a shorter amount of time when the sequence of (heuristic) choices is compatible with its assignment and it requires little back-tracking effort.

6.5 Incremental and Multi-Objective OMT Evaluation

We illustrate the main results of our investigation on single-objective, incremental and multiple-independent OMT($\mathcal{LRA} \cup \mathcal{T}$), that we have previously published in [ST15c]. For a much more detailed overview of this experiment, we refer the interested reader to our publication [ST15c].

Remark 6.5.1. Although the results shown hereafter were obtained with a version of the OMT tools that is now outdated, we have not repeated our experimental evaluation for this Ph.D. dissertation because we have no reason to believe that the results of doing so would lead to different conclusions. To the best of our knowledge, no further advancement in the field of OMT touched any of the involved technologies. Moreover, no new multi-objective OMT tool appeared on the horizon.

Experiment Setup. The benchmarkset consists of 1103 multiple-independent OMT instances $\langle \varphi, \mathcal{O} \rangle_{\Box}$ taken from [LAK+14], where φ is a ground SMT($\mathcal{LRA} \cup \mathcal{T}$) formula, generated starting from a C program belonging to the SW Verification Competition of 2013, and $\mathcal{O}$ is a set $\{\mathsf{obj}_1, ..., \mathsf{obj}_N\}$ of $\mathcal{LRA}$ objectives, each of which corresponds to some variable in the

tool, configuration & encoding	inst.	term.	timeout	time (s.)
OPTIMATHSAT(SINGLE-OBJECTIVE)	1103	1103	0	16161
OPTIMATHSAT(INCREMENTAL)	1103	1103	0	3477
OPTIMATHSAT(MULTI-OBJECTIVE)	1103	1103	0	**901**
Z3(SINGLE-OBJECTIVE)	1103	1101	2	10002
Z3(INCREMENTAL)	1103	1100	3	8683
Z3(MULTI-OBJECTIVE)	1103	1090	13	1761
SYMBA(100)	1103	1091	12	10917
SYMBA(40)+OPT-Z3	1103	1103	0	1128

Table 6.9: Comparison among OPTIMATHSAT, SYMBA and Z3 on SW verification problems in [LAK+14].

C program. The general goal of the optimization search is to find an over-approximation of the feasible domain of these variables, and therefore each objective must be both minimized and maximized. In this experimental evaluation we used the default *testing workbench* that is described in §6, and we set a timeout of $1200s$. on the running time of each OMT solver.

Given a multiple-independent OMT instance $\langle \varphi, \mathsf{obj}_1, ..., \mathsf{obj}_N \rangle_\square$, we considered three different approaches for dealing with it:

- SINGLE-OBJECTIVE: the original OMT instance is split into N independent single-objective OMT problems $\langle \varphi, \mathsf{obj}_i \rangle$; The cumulative time required to sequentially solve all N instances is taken;
- INCREMENTAL: as above, without restarting the OMT solver in-between an optimization search and the other; We leverage the incremental interface and replace each goal obj_i with its successor obj_{i+1} until all goals have been considered; The SMT formula φ is asserted at the beginning of the search and never popped;
- MULTI-OBJECTIVE: the original OMT instance is fed directly to the Multiple-Independent OMT engine described in §4.6.2.

We tested each approach using both Z3 (v. 4.3.3) and OPTIMATHSAT (v. 1.2.1), and compared their performance with SYMBA, using its two best-performing configurations — namely SYMBA(100) and SYMBA(40)+OPT-Z3— that were identified on this benchmark-set in [LAK+14]. In this experimental evaluation the BCLT OMT solver is not considered as it does not support OMT($\mathcal{LRA} \cup \mathcal{T}$).

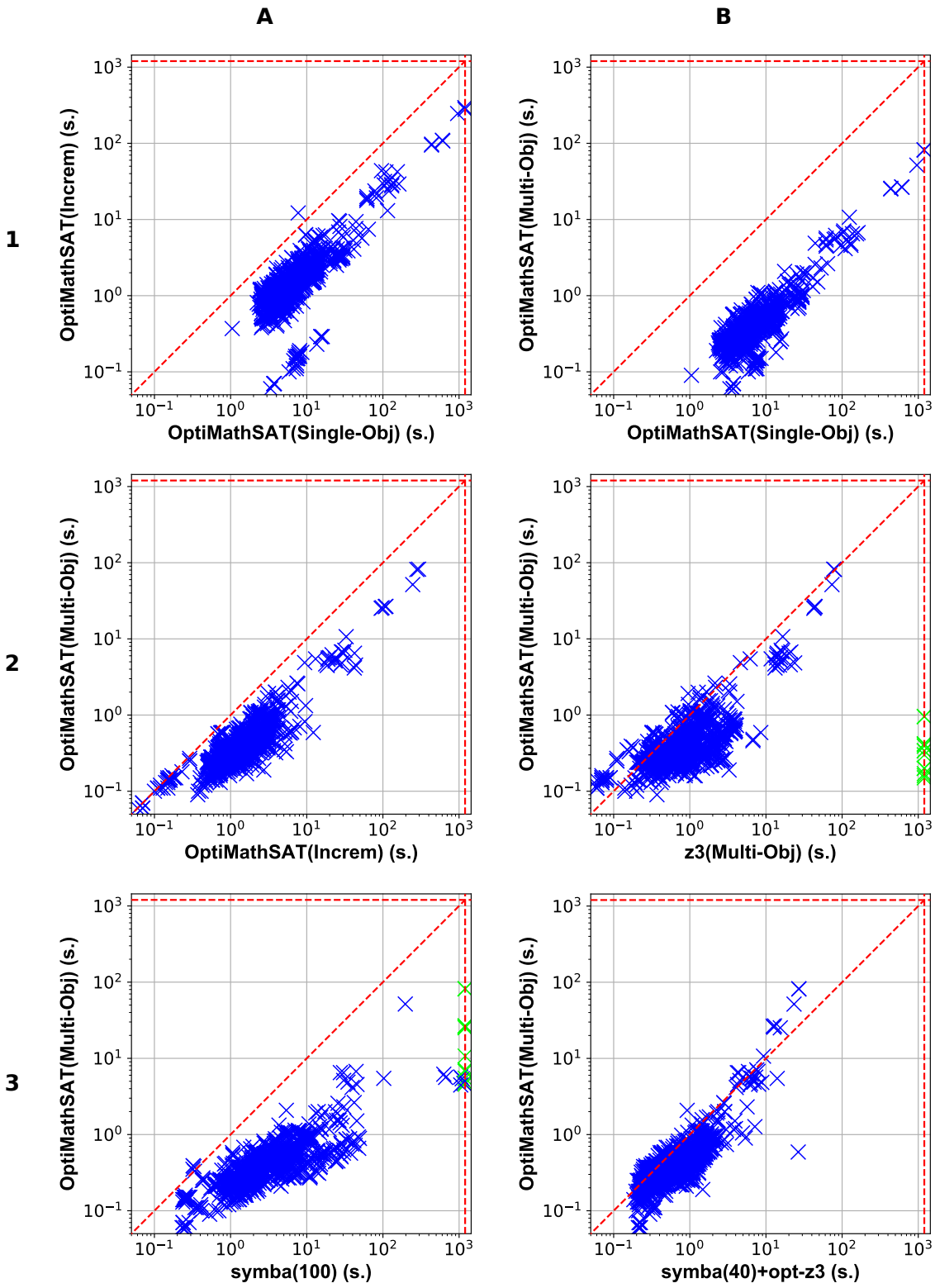


Figure 6.9: Pairwise comparisons on multiple-independent OMT formulas of [LAK+14]. (Blue points denote satisfiable benchmarks, green denotes a timeout.)

Experiment Results. The global performance of each configuration under test is shown in Figure 6.9. Figure 6.9, instead, provides some relevant pairwise comparisons among the configurations being tested.

By looking at first plot in Figure 6.9, we observe that by exploiting the incremental interface of OPTIMATHSAT it is possible to obtain an uniformly relevant speedup. This is explained by the fact that the OMT solver is given a chance to reuse any learned clause in subsequent iterations of the optimization search, which can save a significant amount of effort as described in §4.5.

The second plot of 6.9 shows that a more drastic performance speedup —about one order of magnitude— is obtained when the Multiple-Independent OMT engine is used. The explanation for this performance improvement is twofold. First, at each step of the optimization search the OMT solver can improve the lower and upper bound of multiple objectives at the same time, effectively sharing lots of Boolean and $\mathcal{LIRA}$ search. Second, the OMT solver needs only one *certification* step to prove the absence of an improving solution for all the objectives being considered[33]. The performance improvement obtained by using the Multiple-Independent OMT engine is significantly better than with incremental OMT, as shown in the third plot of Figure 6.9.

Analogous considerations hold for Z3.

The last three scatter plots of Figure 6.9 show the pairwise comparison between OPTIMATHSAT(MULTI-OBJECTIVE) and the following three tools configurations: Z3(MULTI-OBJECTIVE) (plot $2B$), SYMBA(100) (plot $3A$) and SYMBA(40)+OPT-Z3 (plot $3B$). From these plots, we observe that OPTIMATHSAT performs much better than the default configuration of SYMBA, namely SYMBA(100), and significantly better than both SYMBA(40)+OPT-Z3 and Z3(MULTI-OBJECTIVE).

6.6 OMT vs MINIZINC

As illustrated in Section §2.4.2, as part of the work of this Ph.D. we implemented in OPTIMATHSAT a new interface for problems encoded in the MINIZINC format. This extension enables a comparison with the other *Finite Domain Constraint Programming* (FDCP) solvers that support MINIZINC.

In the experimental evaluation in Section §6.6.1, we compare OPTIMATHSAT against other top-scoring MINIZINC solvers on a set of benchmarks coming from the MINIZINC Challenge

[33] As mentioned in §2.3.1, the empirical evidence of [ST12, ST15a, ST15c] has shown that the *certification* step is typically much more expensive than generating a new model when dealing with a $\mathcal{LIRA}$ objective.

2016. In the experiments reported in Section §6.6.2, instead, we show a comparison among OPTIMATHSAT and other MINIZINC tools on benchmark-sets derived from OMT applications.

The goal of these experiments is to investigate the behavior of tools coming from two completely different worlds. In the future, these preliminary results can be used to improve the MINIZINC interface of OPTIMATHSAT and extend its range of applications even further.

6.6.1 MINIZINC Challenge (2016)

Experiment Setup. The benchmark-set is comprised by the 100 MINIZINC formulas used at the MINIZINC Challenge of 2016. In this experiment, we evaluated OPTIMATHSAT (v. 1.6.0) and a selection of the top-scoring MINIZINC solvers of the last years, including CHOCO (v. 4.0.4), CHUFFED, G12(FD) (v. 1.6.0), GECODE (v. 6.0.1), GUROBI (v. 8.0.1), HAIFACSP (v. 1.3.0), JACOP (v. 4.5.0), IZPLUS (v. 3.5.0), OR-TOOLS (v. 6.7.4981) and PICAT (v. 2.4#8). We have been unable to include FZN2SMT [BSV10, BPSV12] in our experimental evaluation because the tool does not seem to be compatible with the new features introduced by MINIZINC v. 2.0, and the benchmark-set is not compatible MINIZINC v. 1.6, which was used by FZN2SMT when it was released. Unless otherwise specified, we used the most recently available version of each MINIZINC tool at the time in which the experiment was performed. Each MINIZINC benchmark used in this experiment was first converted in the FLATZINC format using the MZN2FZN compiler and, whenever available, the directory of global constraints corresponding to the MINIZINC tool under examination. The only exception to this rule is PICAT(CP), for which we have used the standard directory of global constraints, because the vendor-distributed directory of global constraints resulted in some incorrect responses. Each solver was given up to $1200s$. to solve each formula.

We use two different configurations of OPTIMATHSAT, one encoding *Integer* variables in the MINIZINC model with the *Int* sort of SMT-LIBV2, named OPTIMATHSAT(FZN+INT), and the other, called OPTIMATHSAT(FZN+BV), using *Bit-Vectors*. With the former configuration, the optimization search is performed using the OMT($\mathcal{LIRA} \cup \mathcal{T}$) procedures described in Section §4.1. The latter, instead, uses the OBV-BS algorithm described in Section §4.3.3.

Experiment Results. The global results of this experiment are shown in Table 6.10 and in the cactus plot of Figure 6.10. Figures 6.11, 6.12, 6.13, 6.14 and 6.15, instead, provide some relevant pairwise comparisons among the MINIZINC tools and the two OPTIMATHSAT configurations being tested. Using the experimental data, we separately computed the virtual best configuration among all MINIZINC solvers (i.e. VIRTUAL BEST(MINIZINC)) and the one among both OPTIMATHSAT configurations (i.e. VIRTUAL BEST(OPTIMATHSAT)), and also

tool, configuration & encoding	inst.	term.	t.o.	uns.	err.	time (s.)	B1	B2
G12(FD)	100	21	75	0	4	8656	0	4
PICAT(CP)	100	21	79	0	0	8324	0	3
GECODE()	100	39	61	0	0	12163	0	16
CHOCO()	100	47	53	0	0	15679	1	19
IZPLUS()	100	50	50	0	0	6491	7	33
CHUFFED()	100	53	42	5	0	4822	5	34
JACOP()	100	58	42	0	0	16236	0	26
OR-TOOLS(CP)	100	62	8	30	0	1570	27	59
GUROBI()	100	72	28	0	0	5382	15	56
HAIFACSP()	100	73	27	0	0	5035	10	57
PICAT(SAT)	100	74	26	0	0	8637	11	45
OR-TOOLS(SAT)	100	**84**	16	0	0	12163	33	80
VIRTUAL BEST(MINIZINC)	100	93	7	0	0	3657	-	92
OPTIMATHSAT(FZN+INT)	100	55	42	3	0	7112	1	38
OPTIMATHSAT(FZN+BV)	100	**57**	43	0	0	13306	0	30
VIRTUAL BEST(OPTIMATHSAT)	100	68	32	0	0	9564	1	-
VIRTUAL BEST(ALL)	100	93	7	0	0	3657	-	-

Table 6.10: A comparison on the performance of OPTIMATHSAT and various top-scoring MINIZINC solvers on the MINIZINC Challenge 2016 formulas. The columns list the total number of instances (inst.), the number of instances solved (term.), the number of timeouts (t.o.), the number of unsupported problems (uns.), the number of run-time errors (err.), the total solving time for all solved instances (time), the number of instances solved faster or equal than VIRTUAL BEST(MINIZINC) (B1) and the number of instances solved faster or equal than VIRTUAL BEST(OPTIMATHSAT) (B2).

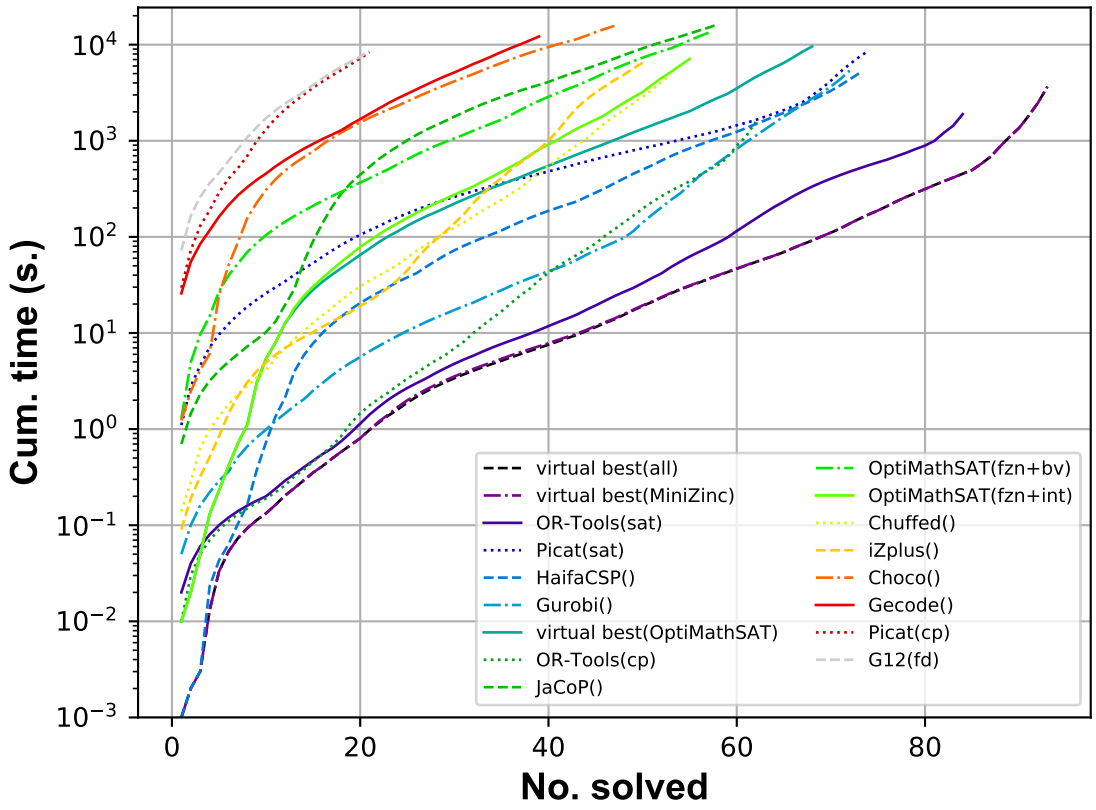


Figure 6.10: A comparison on the performance of OPTIMATHSAT and various top-scoring MINIZINC solvers on the MINIZINC Challenge 2016 formulas.

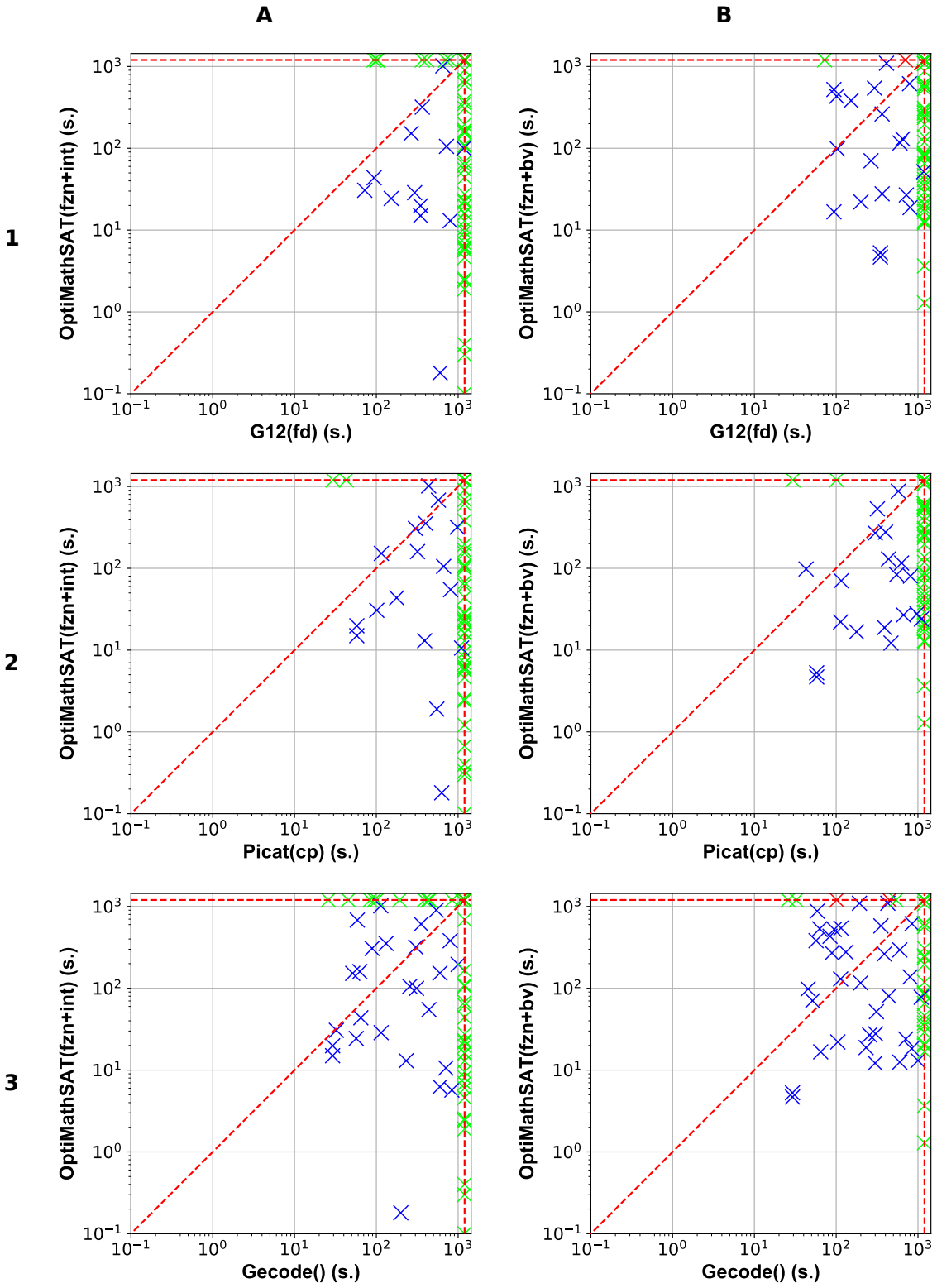


Figure 6.11: Pairwise comparisons on the MINIZINC Challenge 2016 formulas between OPTIMATHSAT, G12(FD), PICAT(CP) and GECODE. (Blue points denote satisfiable benchmarks, green denotes a timeout and red denotes unsupported formulas)

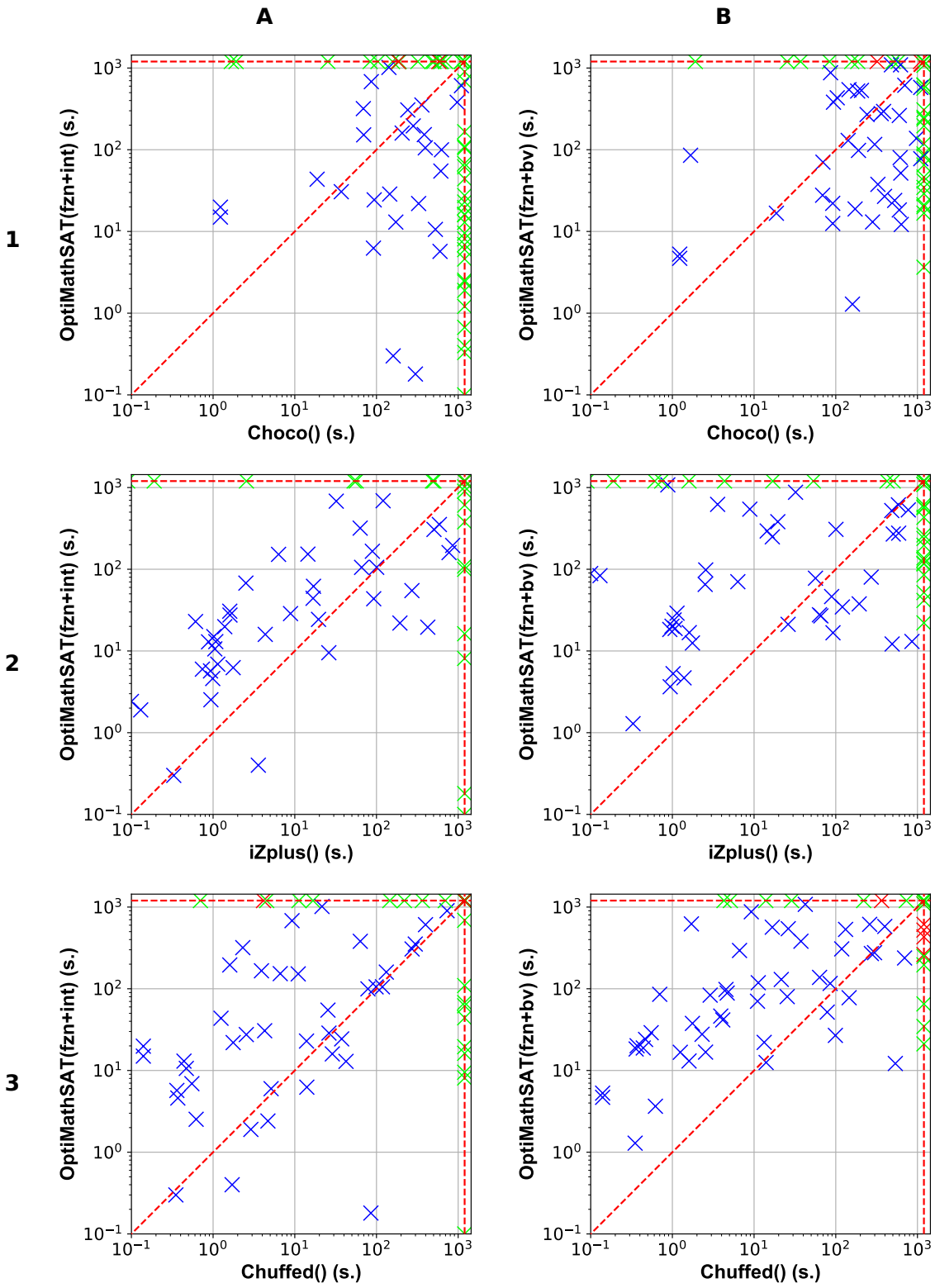


Figure 6.12: Pairwise comparisons on the MINIZINC Challenge 2016 formulas between OPTIMATHSAT, CHOCO, IZPLUS and CHUFFED. (Blue points denote satisfiable benchmarks, green denotes a timeout and red denotes unsupported formulas)

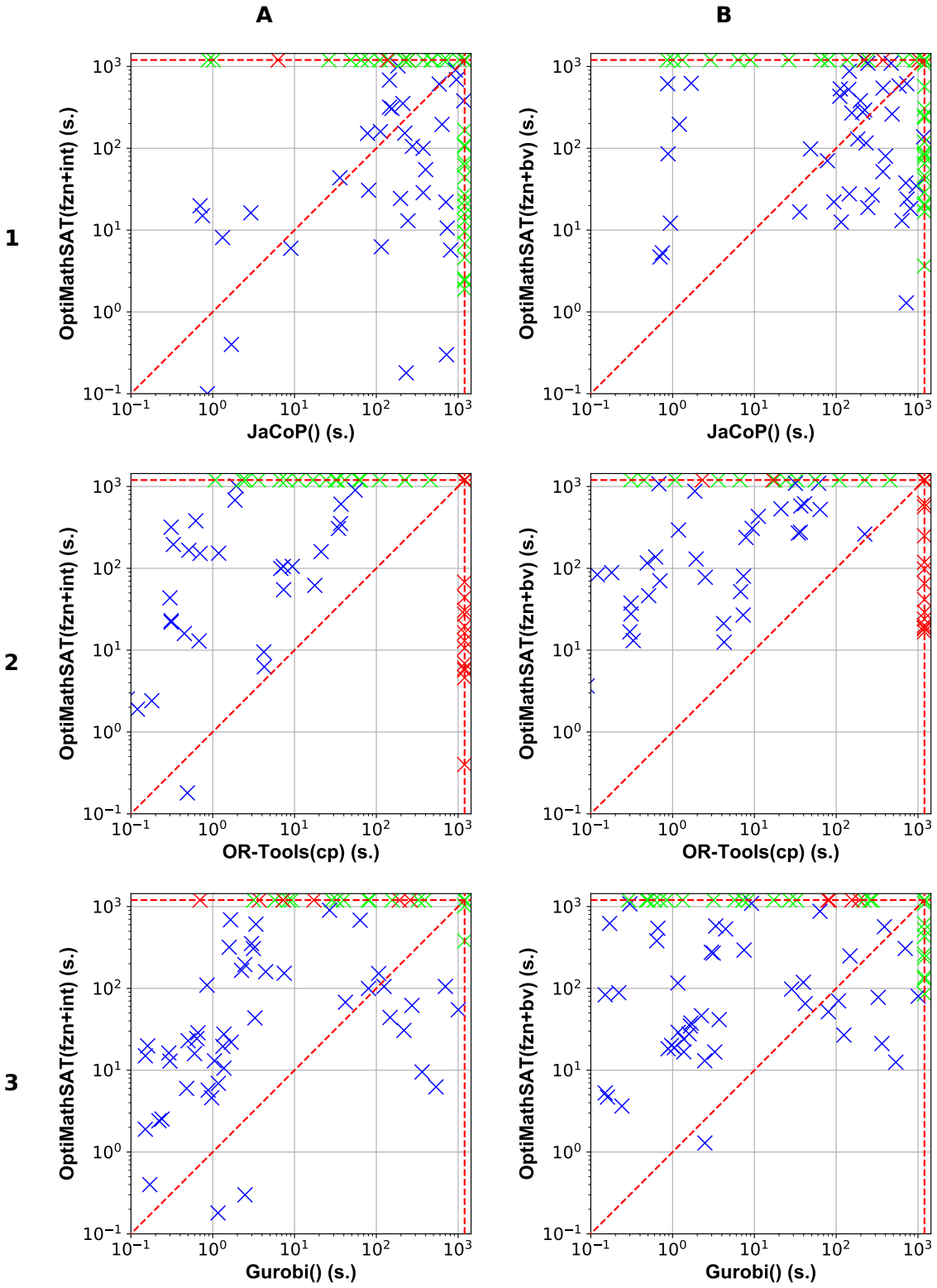


Figure 6.13: Pairwise comparisons on the MINIZINC Challenge 2016 formulas between OPTIMATHSAT, JACOP, OR-TOOLS(CP) and GUROBI. (Blue points denote satisfiable benchmarks, green denotes a timeout and red denotes unsupported formulas)

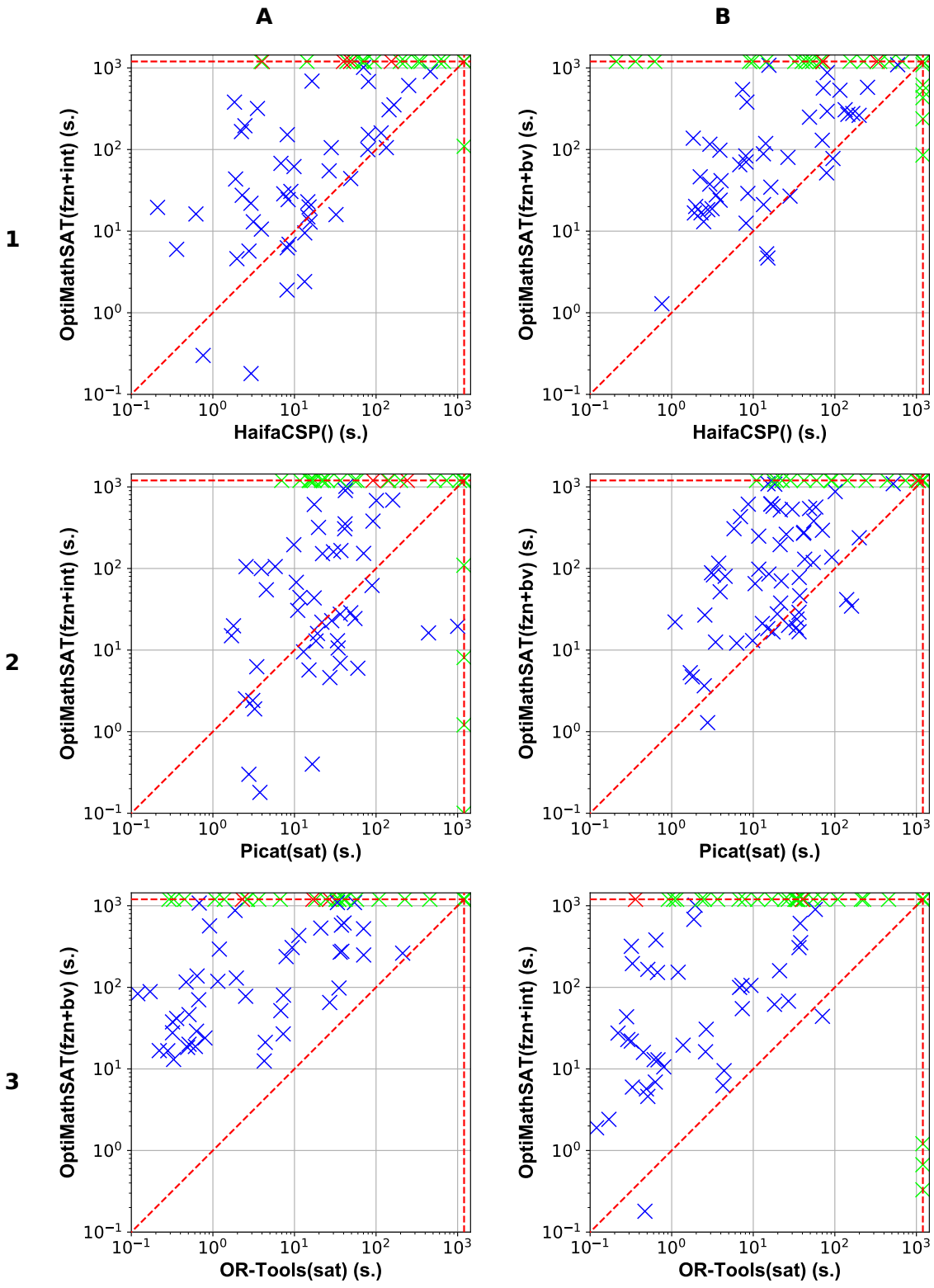


Figure 6.14: Pairwise comparisons on the MINIZINC Challenge 2016 formulas between OPTIMATHSAT, HAIFACSP, PICAT(SAT) and OR-TOOLS(SAT). (Blue points denote satisfiable benchmarks, green denotes a timeout and red denotes unsupported formulas)

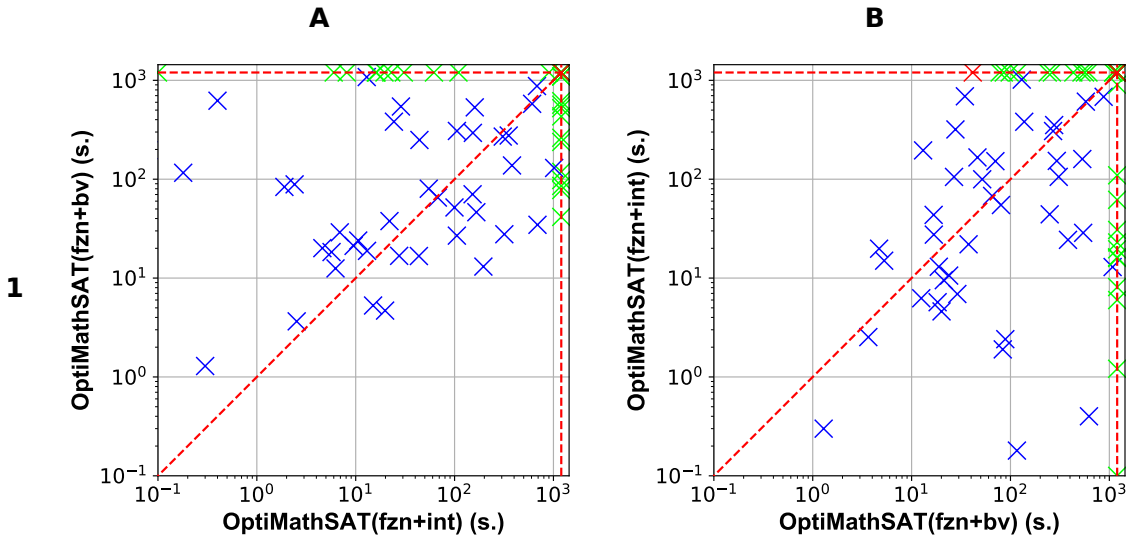


Figure 6.15: Pairwise comparisons on the MINIZINC Challenge 2016 formulas between the two OPTIMATHSAT configurations. (Blue points denote satisfiable benchmarks, green denotes a timeout and red denotes unsupported formulas)

the virtual best among all tools considered in the experiment (i.e. VIRTUAL BEST(ALL)). The last two columns in Table 6.10 list the number of problems for which the given configuration scored better (or equal) than the VIRTUAL BEST(MINIZINC) (col. *B1*) and the VIRTUAL BEST(OPTIMATHSAT) (col. *B2*) configurations.

We start by looking at the MINIZINC solvers in Table 6.10. The best performing tool is OR-TOOLS(SAT), followed by PICAT(SAT), HAIFACSP, GUROBI and OR-TOOLS(CP). By looking at column *B1*, we observe that these MINIZINC solvers dominate over all other MINIZINC solvers considered in this experiment. We observe that this result seems to be compatible with the outcome of the recent (official) editions of the MINIZINC Challenge. The G12(FD) solver encountered 4 `sig-sev` errors, whereas CHUFFED and OR-TOOLS(CP) were unable to handle 5 and 30 instances respectively, due to unsupported constraints in the FLATZINC encoding.

Among the two configurations of OPTIMATHSAT being tested, we observe that there are 3 unsupported problems with OPTIMATHSAT(FZN+INT). This is due to a limitation of the MINIZINC interface that is not able to deal with any, even apparent, non-linear constraint appearing in the input model. The other configuration, called OPTIMATHSAT(FZN+BV), is not affected by this problem and it is able to solve a couple more instances. This achievement, however, comes at the price of a significant overhead, as it can be seen from the cactus plot in Figure 6.10. Interestingly, the pairwise comparison depicted in Figure 6.15, shows that the two OPTIMATHSAT configurations have an orthogonal behavior on several instances experi-

encing a timeout with only one of the two. Among those 68 instances solved with at least one configuration, OPTIMATHSAT solves 38 instances faster with the OPTIMATHSAT(FZN+INT) configuration and 30 instances faster with the $\mathcal{BV}$-based configuration.

On the whole, OPTIMATHSAT(FZN+BV) solves fewer problems than OR-TOOLS(SAT), PICAT(SAT), HAIFACSP, GUROBI, OR-TOOLS(CP), and JACOP, and more instances than G12(FD), PICAT(CP), GECODE, CHOCO, IZPLUS and CHUFFED. By looking at column *B1* and the VIRTUAL BEST() rows, we observe that OPTIMATHSAT does not make any previously unsolvable problem become solvable (within the timeout), and that there is only one instance on which it is able to provide and answer faster than any other MINIZINC tool, by a negligible margin. By looking at column *B2*, we observe that OPTIMATHSAT tends to be slower than the majority of the MINIZINC tools, even when it solves more instances.

We conclude by noting that, further experimentation, with a larger set of benchmarks, is necessary to derive any conclusive evidence indicating on which classes of MINIZINC problems OPTIMATHSAT, and OMT tools in general, can be competitive.

6.6.2 OMT Benchmarks

Full Disclosure. The experimental evaluations contained in this Section have been performed by Francesco Contaldo, a Master Degree student at University of Trento, under the joint supervision of the Ph.D. Candidate and Prof. Roberto Sebastiani.

The student designed and implemented OMT2MZN, a compiler that translates formulas encoded in the *Extended* SMT-LIBV2 *format* for optimization, described in Section §2.4.2, into the MINIZINC language [Minb]. This compiler was used by the F. Contaldo to convert a number of OMT benchmark-sets into the MINIZINC format, and then to perform an extensive experimental evaluation on these benchmarks.

Since this research work is closely related to the content presented in this dissertation, and its outcome is of interest as an assessment of the work in this thesis, we have included in this document a few interesting (preliminary) results, in agreement with the author F. Contaldo. A complete summary of this work —including a description of the OMT2MZN compiler, additional experiments and a more in-depth analysis of the results— is going to be included in the Master Thesis of the student.

Hereafter, we show a comparison among OPTIMATHSAT and other FDCP solvers on formulas derived from typical OMT applications.

General Setup. In each experiment, OPTIMATHSAT (v. 1.5.0) is tested twice. The first time using the original SMT-LIBV2-encoded formula, and the second time with the corresponding

(automatically generated) FLATZINC model. Testing OPTIMATHSAT twice serves two main purposes. First, by checking the results of the OMT solver before and after the automatic transformation of SMT-LIBV2-encoded formulas into FLATZINC models, it makes it possible to discover any intrinsic limitation of this transformation. Second, it can provide some indication of any performance loss (or gain) caused by the encoding transformation, although, admittedly, this might be solver-dependent and not affect other FDCP solvers with the same order of magnitude. Hereafter, we refer to the first configuration as OPTIMATHSAT(SMT), and to the second configuration as OPTIMATHSAT(FZN). Internally, both configurations encode the *Int* type of MINIZINC with the *Integer* sort of SMT-LIBV2 and, whenever necessary, use the OMT($\mathcal{LIRA} \cup \mathcal{T}$) procedure described in Section §4.1 for the optimization search.

The FDCP tools selected for these experiments are: CHOCO(v. 4.0.4), CHUFFED, G12(FD) (v. 1.6.0), GECODE (v. 6.0.1), GUROBI (v. 8.0.1), HAIFACSP (v. 1.3.0), IZPLUS (v. 3.5.0), JACOP (v. 4.5.0), MISTRAL (v. 2.0), OR-TOOLS (v. 6.7.4981) and PICAT (v. 2.4.8). Every FDCP solver under consideration is tested only once, using the default configuration, over the (automatically generated) FLATZINC model.

Each experiment has been performed on the default *testing workbench* described in Chapter §6 and, unless otherwise specified, each tool was given up to $900s.$ to solve each formula. On this regard, we note that the time required by the MZN2FZN compiler to translate the MINIZINC instance into a FLATZINC model has been added to the runtime of the FLATZINC solver. The reason for this is that, the MZN2FZN compiler is sometimes able to find the solution of the input model while flattening it, because it applies simple propagation and inference rules along its transformations. When this is the case, the resulting FLATZINC instance contains an hard-coded solution, and the FDCP solver does not need to perform any search on its own.

Bounded Model Checking

Experiment Setup. The benchmark-set used in this experiment is comprised by a subset of 66 OMT($\mathcal{LRA} \cup \mathcal{T}$) formulas taken from [ST15a], transformed into OMT($\mathcal{LIA} \cup \mathcal{T}$) problems. Each instance encodes a parametric verification problem using using Bounded Model Checking (BMC) of invariants and K-Induction.

Experiment Results. The global results of this experiment are shown in Table 6.11. Figure 6.16, instead, shows the cactus plot comparison among those tools that were able to solve at least one formula, and a couple of pairwise comparisons among the two OPTIMATHSAT configurations being tested and PICAT(SAT).

The best performing tool, on this benchmark-set, is OPTIMATHSAT. Interestingly, the

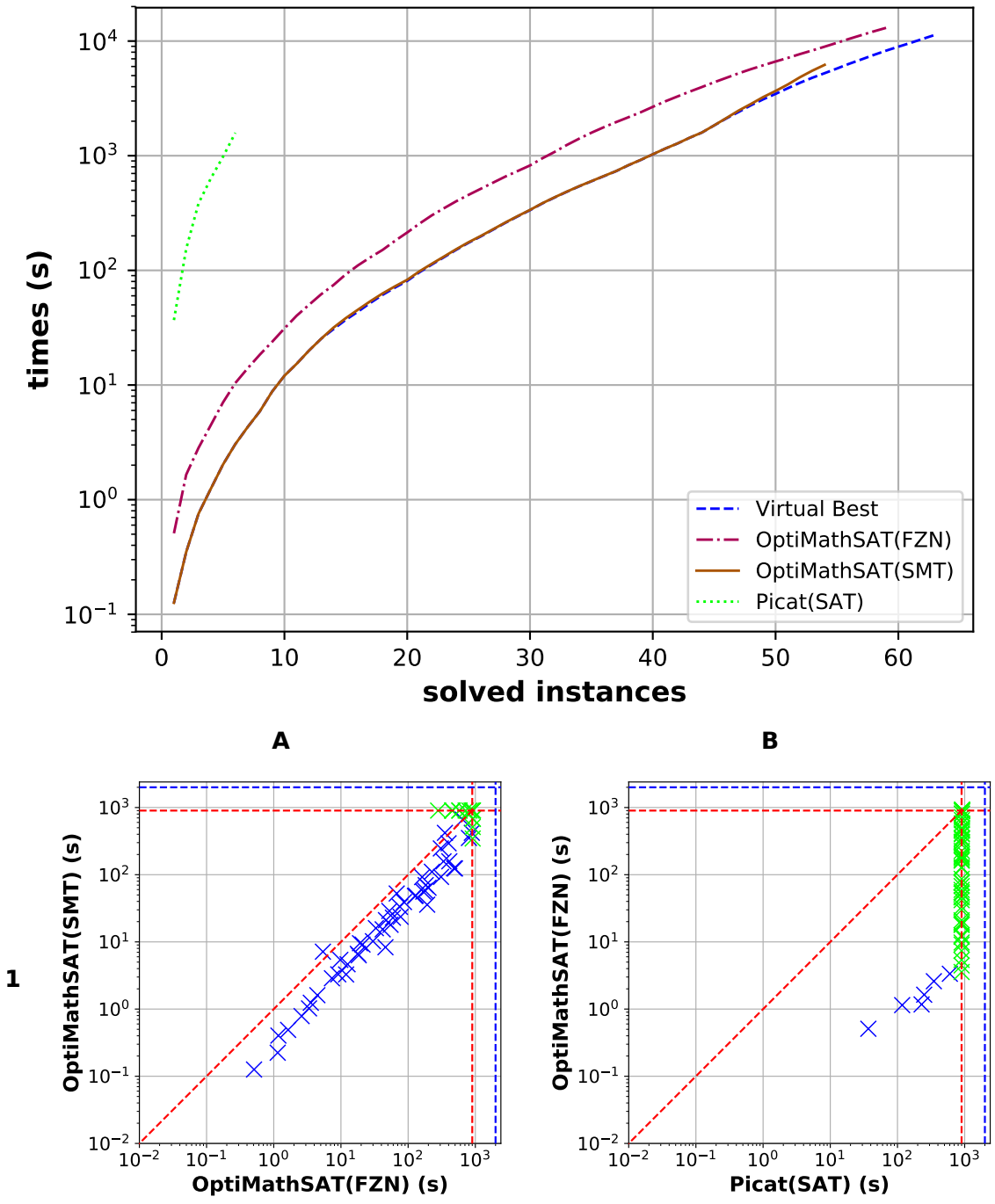


Figure 6.16: Pairwise comparisons on the Bounded Model Checking formulas used in [ST15a], after being converted to OMT($\mathcal{LIA} \cup \mathcal{T}$). (Blue points denote satisfiable benchmarks, green denotes a timeout.)

tool, configuration & encoding	inst.	term.	timeout	unsup.	error	time (s.)
CHOCO()	66	0	66	0	0	0
CHUFFED()	66	0	66	0	0	0
G12(FD)	66	0	0	0	66	0
GECODE()	66	0	66	0	0	0
GUROBI()	66	0	0	66	0	0
HAIFACSP()	66	0	66	0	0	0
IZPLUS()	66	0	0	0	66	0
JACOP()	66	0	62	0	4	0
MISTRAL()	66	0	0	0	66	0
OR-TOOLS(SAT)	66	0	66	0	0	0
OR-TOOLS(CP)	66	0	66	0	0	0
PICAT(SAT)	66	6	60	0	0	1583
PICAT(CP)	66	0	66	0	0	0
OPTIMATHSAT(FZN)	66	**59**	7	0	0	13038
OPTIMATHSAT(SMT)	66	54	12	0	0	6204

Table 6.11: A comparison on the performance of OPTIMATHSAT and various top-scoring FDCP solvers on a subset of the Bounded Model Checking formulas used in [ST15a], after being converted to OMT($\mathcal{LIA} \cup \mathcal{T}$).

OMT solver is able to solve 5 more instances when the input formulas are encoded in the FLATZINC format. By looking at Figure 6.16, we can see that the performance of OPTIMATHSAT(SMT) are very close to the *virtual best* configuration, except for the few more instances that OPTIMATHSAT(FZN) is able to solve.

Except for PICAT(SAT), that terminates on 6 instances, all other FDCP solvers evaluated in this experiment are unable to solve any formula in the benchmark-set. For GUROBI, the problem is caused by the MZN2FZN-GUROBI compiler, which encounters an error while converting the MINIZINC model generated with OMT2MZN tool. The 66 errors of G12(FD) and the 4 of JACOP are due to hitting an upper bound to the program's available memory. The errors of IZPLUS and MISTRAL, instead, are caused by an invalid execution ending with a *core dump*.

Learning Modulo Theories (LMT) with OMT($\mathcal{PB} \cup \mathcal{T}$)

Experiment Setup. This experiment considers 510 OMT($\mathcal{PB} \cup \mathcal{T}$) formulas generated by PYLMT [pyl], a tool for Structured Learning Modulo Theories [TSP17] that uses OPTIMATHSAT as back-end engine for doing inference in the context of machine learning in hybrid do-

mains. By construction, every instance in this benchmark-set is satisfiable. We refer the reader to the description of the "LMT with OMT($\mathcal{PB} \cup \mathcal{T}$)" experiment in Section §6.2.4 for more details on the benchmark-set.

The FDCP solvers evaluated in this experimental evaluation are G12(MIP), GECODE and GUROBI. The remaining tools were discarded due to missing or incomplete support for floating-point arithmetic.

Experiment Results. The results of this experiment, including a cactus plot comparison, are shown in Table 6.12. In this table, we have added a new column named "corr." that indicates the number of formulas for which the solvers returned a correct solution. A solution is considered correct if it is not equal to UNSAT and the relative error Δ is smaller or equal 10^{-6}; Δ is defined as follows:

$$\Delta \stackrel{\text{def}}{=} \frac{|o_{smt2} - o_{fzn}|}{|o_{smt2}|} \tag{6.3}$$

where o_{smt2} is the optimum value of the objective function found by OPTIMATHSAT(SMT), used as a reference, and o_{fzn} is the optimum value found by the other solver. We recall here that OPTIMATHSAT uses infinite precision arithmetic reasoning, and we also note that the o_{smt2} value has been independently verified both for correctness and optimality.

From the table, it can be seen that OPTIMATHSAT terminates on 502 SMT-LIBV2-encoded formulas in the benchmark-set, with the correct solution. However, when the OMT solver is given as input the (automatically generated) FLATZINC models, it solves 236 problems within the timeout and for only 35 of these it returns the correct solution. The outcome for the other FDCP solvers is similar. For instance, GUROBI terminates on 506 instances but for only 22 of these it provides the correct solution.

We explain the incorrect result as being the direct consequence of the intrinsic limitations of the FLATZINC language, that does not provide any datatype with the same characteristics of the *Real* sort of SMT-LIBV2. Therefore, during the translation of the original SMT-LIBV2 formulas in the FLATZINC format, infinite-precision *Real* values get replaced by finite-precision Floating-Point constants. A subsequent analysis revealed that these approximations are introduced by the MZN2FZN compiler.

tool, configuration & encoding	inst.	term.	corr.	timeout	error	time (s.)
G12(FD)	510	436	0	71	3	27568
GECODE()	510	212	24	298	0	145
GUROBI()	510	506	22	4	0	4163
OPTIMATHSAT(FZN)	510	236	**35**	274	0	205
OPTIMATHSAT(SMT)	510	502	**502**	8	0	4740

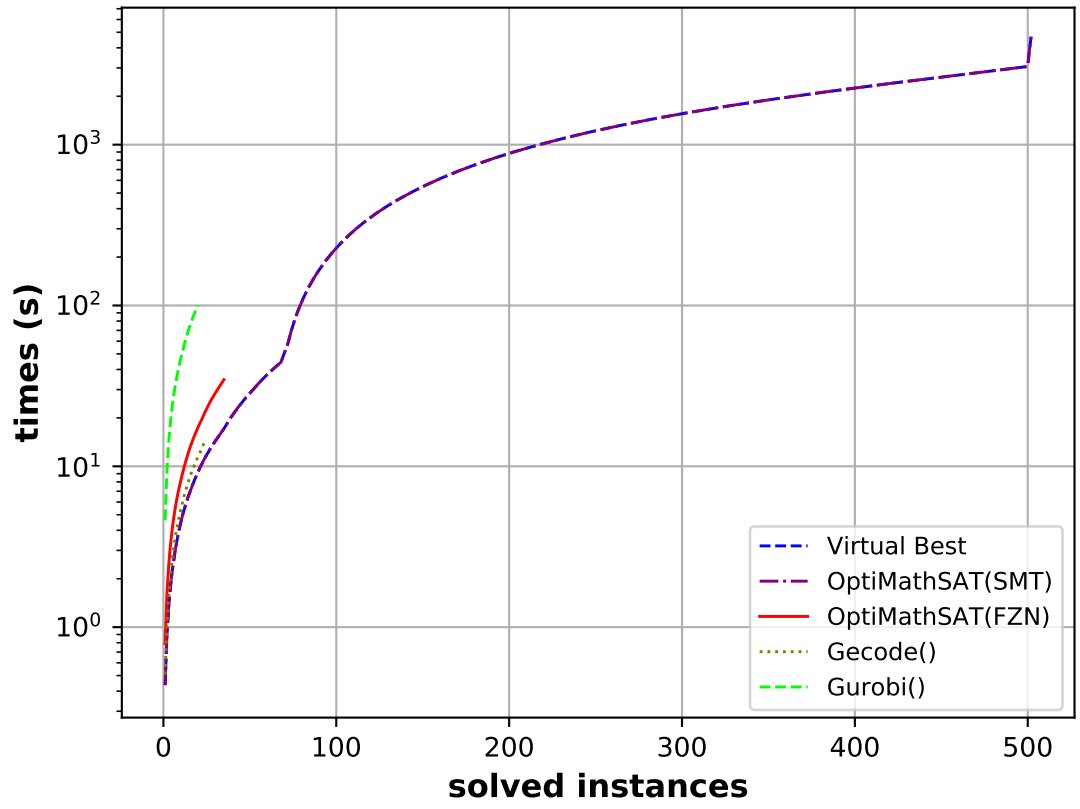


Table 6.12: A comparison on the performance of OPTIMATHSAT and various FDCP solvers on Learning Modulo Theories (LMT) formulas from [TSP17]. The cactus plot does not include any instance terminated with an incorrect result.

Chapter 7

Applications

In comparison with the well-established fields of *Satisfiability Modulo Theories* (§2.2) and even of *(Finite Domain) Constraints Programming with Optimization* (§2.4), outside the context of MAXSMT the research on *Optimization Modulo Theories* is still at an early stage and it could benefit from more attention to develop more sophisticated and efficient techniques for its applications.

This chapter contains a collection of scientific works using *Optimization Modulo Theories* for both research purposes and real-world applications. We split our presentation in two parts. Section §7.1 contains a selection of recent scientific works using *Optimization Modulo Theories* at their core, not necessarily focused on OPTIMATHSAT. Section §7.2, instead, describes a few publications dealing with real-world applications that have been of primary importance to guide both the development of OPTIMATHSAT and the research advances presented in this dissertation.

The goals of this chapter are not only to justify our research work on *Optimization Modulo Theories* and to provide evidence of its largely untapped potential, but also to draw the attention of the community on this research topic and the potential benefits of further improvements.

7.1 OMT Applications

Hereafter, we briefly describe a selection of scientific studies using *Optimization Modulo Theories* for some relevant applications in the domains of formal verification, program analysis, scheduling, planning with resources and more. On the whole, our goal is to provide —with a few examples— concrete evidence of the great potential of *Optimization Modulo Theories* when dealing with real-world applications of primary interest. For this reason, we omit several details from the scientific publications being cited, and limit our view to those features of the

target application that have been technologically enabled, or enriched, by OMT.

Static Analysis

Static analysis deals with the problem of automatically deriving universally valid properties of a (piece of) program such as the fact that a given error state is unreachable in every possible execution. In the last years, *Satisfiability Modulo Theories* has been increasingly adopted among static analysis tools, thanks to advances both in terms of expressiveness and in terms of performance.

In [CLO+16], *Optimization Modulo Theories*, has been used to speedup the *constraint-based method*, a well-known approach that has been applied to a plethora of tasks in the context of program analysis such as invariant generation and proving termination. Typically, the *constraint-based method* requires the capability to deal with non-linear SMT formulas to instantiate constraints with Farkas' Lemma. In their work, Candeago et al. proposed a new encoding of this problem that is applicable when the program at hand can be restricted to the Difference Logic ($\mathcal{DL}$) fragment of linear arithmetic. This approach is based on a MAXSMT formulation and does not require any capability to reason on non-linear arithmetic. As a result, it allows the use of much cheaper SMT solving techniques and a wider range of (MAXSMT) tools to deal with the same problem. The paper is complemented with an experimental evaluation showing the high impact of this approach on the performance of the VERYMAX verification system, that uses the OMT solver BCLT at its core.

In [Kar17], Karpenkov applies *Optimization Modulo Theories* to the problem of finding inductive invariants, that is, universally valid properties that hold in the initial state of a program and imply themselves under the program transition. To this aim, the novel policy iteration approach presented in this dissertation, called *Local Policy Iteration* (LPI), uses OMT as part of a widening operator that is guaranteed to return the least inductive invariant after finitely many applications. The LPI approach is implemented inside the open-source software verifier CPACHECKER, [BK11], and it is uses the OMT solver Z3 as a black-box via the JAVASMT API [KFB16].

Formal Verification & Model Checking

In the context of bounded program verification, functional properties are checked by analyzing a finite number of objects and loop iterations. Traditionally, the extent of this verification is controlled by the end-user, that provides both the number of objects and the number of loop iterations to consider. In [LTBT17], Liu et al. proposed a new approach for automatically computing the exact number of loop iterations to be checked based on the number of objects being

considered. For this task, the input program and its specification are encoded as an *Optimization Modulo Theories* formula in which the objective function represents the number of loop iterations. The authors implement their approach in the prototype tool BOUNDJ, using the OMT solver Z3 as a black-box.

Recently, *Optimization Modulo Theories* has also been used in the context of formal verification of hybrid dynamical systems to synthesize barrier certificates [Rat17]. A barrier certificate can be viewed as a function of the state of the dynamic system that separates an unsafe region from all system trajectories starting from a given set of initial conditions. For this reason, it can be used to prove the safety of the system itself. In [Rat17], a candidate barrier certificate is generated from a simulation of the dynamic system and subsequently improved via a refinement loop in the case in which it does not meet the requirements for being a barrier certificate, i.e. it is *spurious*. In this framework, the role of *Optimization Modulo Theories* is to compute the candidate barrier certificate that ensures the greatest amount of progress at each step of the algorithm. The paper is complemented with an implementation of the algorithm, using OPTIMATHSAT as OMT oracle, and an experimental evaluation showcasing the advertised functionality.

Scheduling and Planning with Resources

In [KBE17, KEB18, KBE18], Kovásznai et al. investigated the use of *Optimization Modulo Theories* to deal with Wireless Sensor Networks (WSNs). In a Wireless Sensor Network (WSN), dozens or even hundreds of spatially distributed sensor and actuator nodes cooperate with one another to achieve a common goal like, e.g., monitoring physical or environmental conditions and react upon them. Typically, WSNs are deployed with limited or no access to an external source of power, so that batteries still remain the most prevalent power supply currently used. Therefore, it is of critical importance to find the sleep/wake-up scheduling that is optimal in terms of energy efficiency, and to maximize the amount of time for which a WSN can dependably deliver its services while enduring, at the same time, the loss of nodes in the network. In [KBE17], Kovásznai proposed a formalization of this problem in *Optimization Modulo Theories*, and compared the performance of Z3, SYMBA and OPTIMATHSAT on a set of OMT($\mathcal{QF\_UFLIA}$) formulas encoding randomly generated WSNs. This study was further extended in [KEB18], where the same authors investigated which dependability and safety constraints appear to be the most challenging when dealing with increasingly denser WSNs. In this paper, the authors focused their efforts on the OMT solver OPTIMATHSAT.

In [LÁN+17, LAN+18], Leofante et al. leveraged the expressiveness of *Optimization Modulo Theories* to deal with various multi-robot scheduling problems. Typically, this class of problems is handled with heuristic approaches that, however, do not guarantee the optimality

of the computed solution. The integrated system presented in [LÁN+17, LAN+18] employs the OMT solver Z3 to synthesize optimal plans for multi-robot systems, and an *on-line executive* to deploy the generated plan and collect runtime feedback on its execution by the fleet of robots. The authors thoroughly investigated the effectiveness of this approach on the *RoboCup Logistics League* (RCLL) problem, demonstrating its capabilities and weaknesses. The experience gained in [LÁN+17, LAN+18] has been leveraged by Bit-Monnot et al. to design SMARTPLAN, [BLP+18], a task planner for smart factories based on *Optimization Modulo Theories*. The tool extends the previous work with a new general-purpose OMT-based encoding that can deal with logistic applications outside the multi-robot scenario presented in [LÁN+17, LAN+18].

A common problem when deploying automated production systems is to find the optimal assignment of jobs to machines. A well-known NP-hard variant of this task is the Job-Shop Scheduling Problem (JSP). Traditionally, JSP is solved using either *Mixed Integer Linear Programming* (MILP), *Constraint Programming* (CP) or ad hoc heuristic algorithms. In [RBÅ18], Roselli et al. considered a formulation of the Job-Shop Scheduling Problem based on *Optimization Modulo Theories*. In their work, the authors compared the performance of GUROBI, a MILP solver, with those of the OMT solvers Z3 and OPTIMATHSAT. The results of their experimental evaluation show that OMT, in particular when using Z3, can be a competitive alternative to state-of-the-art commercial MILP solvers both in terms of expressiveness and in terms of solving-time performance.

In a recent paper, [OCS18], Oliver et al. describe an application of *Optimization Modulo Theories* in the context of Time Sensitive Networks (TSN). In particular, in the paper they address the problem of synthesizing communication schedules for the Gate Control Lists (GCLs) defined in the IEEE 802.1Qbv Standard. The synthesis problem, formalized in terms of the Theory of Arrays ($\mathcal{AR}$), leverages OMT to find the realization that minimizes the receiving jitter for the incoming streams of data, and also to find the best trade-off among the effective communication jitter and the number of windows available for each egress port. The experimental results demonstrate the applicability of OMT-based techniques to this approach for small- and medium-sized networks, whereas for larger networks the authors suggest that a combination of heuristic and OMT-based techniques could be the most promising approach.

Software Security Engineering

In many organizations, authorization policies can impose constraints on the workflow of an organization due to security or privacy concerns. These policies are designed to avoid errors and frauds, but they may also represent an obstacle to the completion of certain tasks when these

cannot be executed without violating certain constraints. A violation can be resolved by extending the privileges of some users, or by canceling the execution of a task and, perhaps, missing or changing a previously set goal. From an organizational point of view, the ideal solution is one that minimizes the cost effect of these violations on the business of the organization. In [BdSR18], Bertolissi et al. presented the *Multi-Objective Workflow Satisfiability Problem* (MO-WSP), that formalizes the above problem using Bounded Model Checking and *Optimization Modulo Theories*. The proposed solution has also been validated by the authors by testing it on real-world workflows, and tested for scalability on artificially generated instances.

7.2 OPTIMATHSAT Applications

In the following, we briefly mention a few examples of recent applications —that are very innovative in their respective domains— that have been technologically enabled by Optimization Modulo Theories and that use OPTIMATHSAT as main back-end engine for automated reasoning.

Some of these applications were of primary importance to both push and guide the development of some of the ideas and approaches described in Chapters §4 and §5.

7.2.1 Learning Modulo Theories

In the context of Machine Learning applications, performing inference and learning in hybrid domains is a particularly daunting task, that requires the capability of reasoning over both continuous and Boolean/discrete variables.

In *Structured Learning Modulo Theories* (SLMT), [TSP17], this problem is address by combining (Structured-Output) Support Vector Machines (SVNs) with Optimization Modulo Theories. In the same paper, Teso et al. presented LMT, the first machine learning tool implementing the SLMT approach [lmt]. The software uses OPTIMATHSAT as back-end OMT engine, where it is used as inference and separation oracle.

Figure 7.1 depicts an application example taken from [TSP17], whereby LMT deals with the problem of automatic character drawing. In this application, the SLMT framework uses OPTIMATHSAT to deal with OMT formulas whereby the objective function is comprised by the complex arithmetic combination of several Pseudo-Boolean terms such as the Equation (6.1) reported in Section §6.2.

We refer the reader to [TSP17, Pas16] for details.

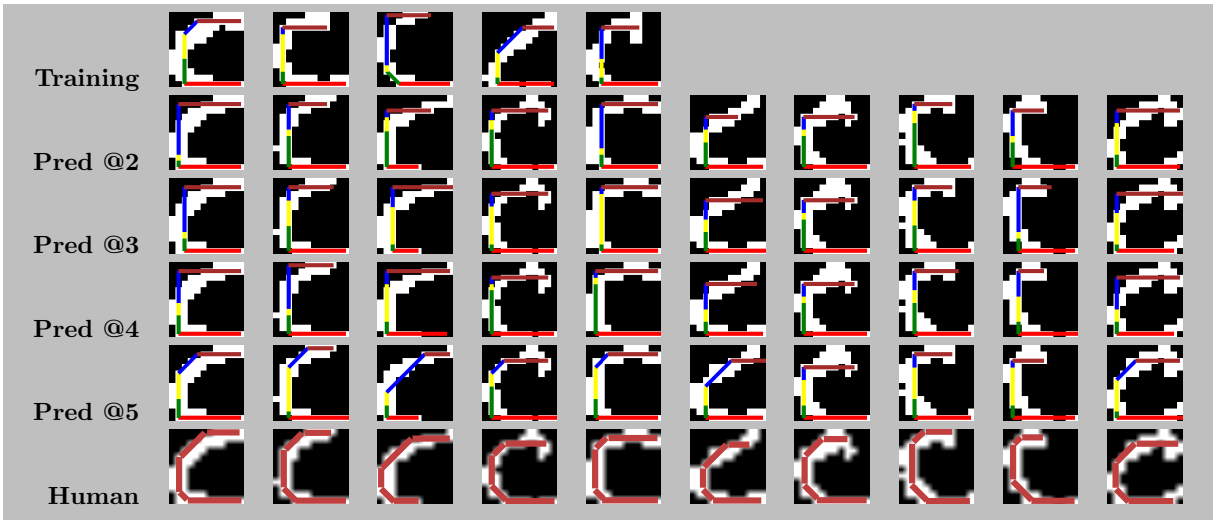


Figure 7.1: An example of SLMT application to the hybrid domain of automatic character drawing taken from [TSP17]. The picture depicts the results for the task of drawing the "C" 12×12-pixel character.

7.2.2 Constrained Goal Models

In the context of Requirements Engineering, Goal Models (GM) are commonly used to represent software requirements, goals and design qualities [VL01].

Constrained Goal Models (CGMs), [NSGM16a, NSGM16b, NSGM17], enriches Goal Models with constraints that can be used to express preferences, numerical attributes and resources (like, e.g., scores, financial cost, workforce, etc.). In [NSGM16a, NSGM16b], the authors presented CGM-TOOL, [cgm], a software that allows for modeling and reasoning on CGMs. Internally, the CGM-TOOL uses the OMT solver OPTIMATHSAT as workhorse engine to automatically verify whether CGMs are realizable and, in such case, find their optimal realizations according to some criteria.

Figure 7.2 show a simple example of a CGM for scheduling a meeting, taken from [NSGM16a, NSGM16b, NSGM17]. We refer the interested reader to [NSGM16a, NSGM16b] for more details and to [ADB+18, AAMF18] for some application examples.

7.2.3 WCET

In the context of real-time systems, it is often useful to compute an upper bound on the worst-case execution time (WCET) of programs subject to hard time constraints. In [HAMM14], Henry et al. described a novel approach for computing the WCET of loop-free C programs with Optimization Modulo Theories. Using OMT, the authors were able to take into account the

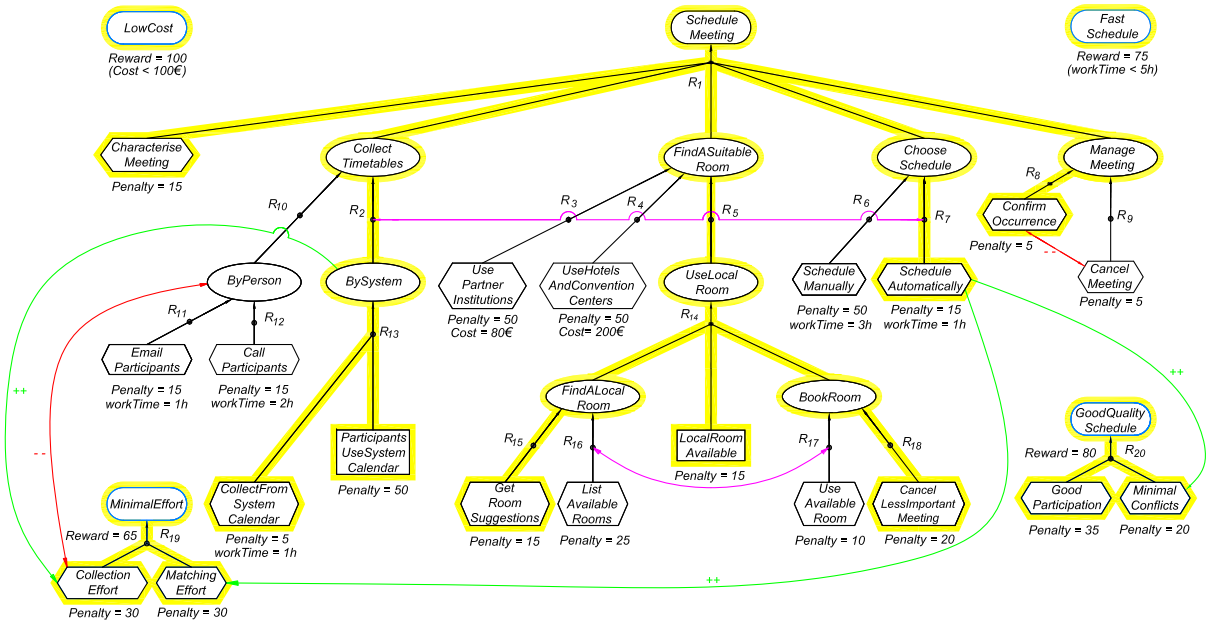


Figure 7.2: A simple CGM example taken from [NSGM16a]. The model highlights, in yellow color, the optimal realization that lexicographically minimizes the sorted list of numerical objectives $\langle Penalty - Reward, workTime, cost \rangle$, found with OPTIMATHSAT.

semantics of a piece of program to prune infeasible paths from the search space and produce a tighter estimate of the WCET, in some cases with an impressive improvement over the state of the art. In their paper, Henry et al. used the OMT solver Z3 and enriched the encoding of the problem with appropriate cuts to bring it to a tractable size.

Recently, the Ph.D. candidate —in collaboration with David Monniaux[34] and Prof. Roberto Sebastiani[35]— reproduced the research work of [HAMM14], and modified it to use OPTIMATHSAT as its main workhorse engine. To achieve this goal, the Ph.D. candidate implemented from scratch a Difference Logic (DL) solver in OPTIMATHSAT. This solver was then extended with the capability of learning $\mathcal{T}$-lemmas derived from subsequent $\mathcal{T}$-conflicts that act as a replacement for the additional cuts used in [HAMM14]. Unpublished preliminary experimental results show the usefulness of this approach. The source code for this project is available at [wce].

[34]David Monniaux, David.Monniaux AT univ-grenoble-alpes.fr, VERIMAG, Grenoble, France.
[35]Prof. Roberto Sebastiani, roberto.sebastiani@unitn.it, DISI, University of Trento, Italy.

7.2.4 Quantum Annealing

In the context of quantum computing, Quantum Annealers (QA) are specialized quantum computers that minimize objective functions, defined over binary variables, by physically exploiting quantum effects [JAG+11, BCI+14]. Currently, Quantum Annealing platforms such as the D-Wave 2000Q [dwa] allow for the optimization of quadratic objectives defined over binary variables (qubits), solving quadratic unconstrained binary optimization (QUBO) problems. In the last decade, QA systems have scaled with Moore-like growth, such that current architectures provide 2048 sparsely-connected qubits, and continued exponential growth is anticipated.

In [BCM+17, BCM+18], Bian et al. have investigated the problem of effectively encoding SAT and MaxSAT problems into QUBOs, so that they can be fed to and solved by state-of-the-art D-Wave 2000Q QAs. To this aim, OPTIMATHSAT is used off-line to create libraries of QUBO encodings of useful Boolean functions by automatically computing an optimal choice of (I) the QUBO input parameter values and of (II) variable-to-qubit placement, so that to maximize a parameter (gap) stating the robustness of the system with respect to noise. The two problems (I) and (I)+(II) are addressed by solving an OMT($\mathcal{LRA}$) and an OMT($\mathcal{LRIA} \cup \mathcal{UF}$) problem respectively. We refer the reader to [BCM+17, BCM+18] for details.

Chapter 8

Conclusions and Future Research Directions

A few years ago, the advent of Optimization Modulo Theories has sprang new opportunities for research and real-world applications, even outside the domains of Formal Verification and Automated Reasoning. Because OMT solving is a resource demanding task, there has been an ever-increasing pressure for more efficient OMT techniques as this technology has become more widespread. At the same time, there has also been an increasing demand for more expressiveness and functionalities that goes way beyond its initial conception.

In this thesis, we have advanced the research on Optimization Modulo Theories along several research directions, focusing on improving its efficiency and expressiveness. We have also presented OPTIMATHSAT, a state-of-the-art Optimization Modulo Theories solver that we have extended to support, among other things, both single- and multi-objective optimization over arbitrary sets of $\mathcal{LRA}$, $\mathcal{LIA}$, $\mathcal{LRIA}$, $\mathcal{BV}$, $\mathcal{FP}$, $\mathcal{PB}$ and MAXSMT cost functions. We have provided a detailed description of OPTIMATHSAT's architecture, of its input/output interfaces and also of the algorithms it implements under the surface. We have concluded our presentation with a selected review of experimental results, partly published in previous papers, which confirm the validity of our implementation.

We believe that the work presented in this thesis builds solid foundations for future research, and it consolidates the position of OPTIMATHSAT as a state-of-the-art Optimization Modulo Theories solver and as a viable alternative to other, competing, tools through a number of small, but significant, improvements.

Nonetheless, the field of Optimization Modulo Theories still remains a largely unexplored territory, with great margins for improvement. Looking at the research presented in this thesis, and focusing on OPTIMATHSAT in particular, we identify the following research goals as main

improvement directions.

First, we envisage the opportunity to explore the possibility to integrate more robust Pareto optimization approaches inside OPTIMATHSAT that perform some kind of systematic exploration search granting a better view of the Pareto front being constructed in the case of $\mathcal{LRA}$ and unbounded objectives (like, e.g., [LTZ05, LLGCM10]).

Second, we envisage the opportunity to investigate —and to integrate into OPTIMATHSAT if successful— OMT versions of the novel SMT procedures for $\mathcal{NLRA}$ [CGI+17a] and for $\mathcal{NLRA}$ plus transcendental functions [CGI+17b], both of which have been recently implemented on top of MATHSAT5.

Third, we envisage the opportunity to further investigate the potential use of Optimization Modulo Theories as an alternative to MINIZINC tools, and extend OPTIMATHSAT to make it the bridge of two worlds that are now seemingly distinctly separated. This goal may require a more in-depth investigation on the strengths and weaknesses of Optimization Modulo Theories with respect to tools coming from the MINIZINC world to identify any technological gap that is critical to the performance. Currently, without such an investigation, we identify three main improvement directions for OPTIMATHSAT: the inclusion of a $\mathcal{T}$-solver for the theory of (finite) sets, that are heavily used in the context of MINIZINC solving, and the development of dedicated procedures for dealing with global constraints.

Finally, we hope for a tighter integration between OPTIMATHSAT and MATHSAT5, that could strengthen the position of both solvers and speed up their adoption.

Bibliography

[AAdB+17] Higo F. Albuquerque, Rodrigo F. Araujo, Iury Valente de Bessa, Lucas C. Cordeiro, and Eddie Batista de Lima Filho. OptCE: A Counterexample-Guided Inductive Optimization Solver. In *SBMF*, volume 10623 of *Lecture Notes in Computer Science*, pages 125–141. Springer, 2017.

[AAdB+18] Rodrigo F. Araujo, Higo F. Albuquerque, Iury Valente de Bessa, Lucas C. Cordeiro, and João Edgar Chaves Filho. Counterexample guided inductive optimization based on satisfiability modulo theories. *Sci. Comput. Program.*, 165:3–23, 2018.

[AAMF18] Nikolaos Argyropoulos, Konstantinos Angelopoulos, Haralambos Mouratidis, and Andrew Fish. Risk-aware decision support with constrained goal models. *Inf. & Comput. Security*, 26(4):472–490, 2018.

[ABCF16] R. Araújo, I. Bessa, L. C. Cordeiro, and J. E. C. Filho. SMT-based Verification Applied to Non-convex Optimization Problems. In *2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 1–8, Nov 2016.

[ABCS05] Gilles Audemard, Marco Bozzano, Alessandro Cimatti, and Roberto Sebastiani. Verifying Industrial Hybrid Systems with MathSAT. In *Proc. BMC 2004*, volume 119 of *ENTCS*. Elsevier, 2005.

[ABKW08] Tobias Achterberg, Timo Berthold, Thorsten Koch, and Kati Wolter. Constraint integer programming: a new approach to integrate CP and MIP. In *Proc. CPAIOR'08*, LNCS, pages 6–20. Springer, 2008.

[ABL09] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial maxsat through satisfiability testing. In Kullmann [Kul09], pages 427–440.

[ABP+11a] Carlos Ansótegui, Miquel Bofill, Miquel Palahı, Josep Suy, and Mateu Villaret. A proposal for solving weighted CSPs with SMT. In *Proceedings of the 10th in-*

ternational workshop on constraint modelling and reformulation (ModRef 2011), pages 5–19, 2011.

[ABP+11b] Carlos Ansótegui, Miquel Bofill, Miquel Palahí, Josep Suy, and Mateu Villaret. Satisfiability Modulo Theories: An Efficient Approach for the Resource-Constrained Project Scheduling Problem. In *SARA*, 2011.

[ABP+11c] Carlos Ansótegui, Miquel Bofill, Miquel Palahı, Josep Suy, and Mateu Villaret. W-MiniZinc: A proposal for modeling weighted CSPs with MiniZinc. In *Proceedings of the 1st International Workshop on MiniZinc (MZN 2011)*, 2011.

[ABP+13] Carlos Ansótegui, Miquel Bofill, Miquel Palahí, Josep Suy, and Mateu Villaret. Solving weighted CSPs with meta-constraints by reformulation into Satisfiability Modulo Theories. *Constraints*, 18(2):236–268, 2013.

[Ach09] Tobias Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 07 2009.

[ACKS02] G. Audemard, A. Cimatti, A. Korniłowicz, and R. Sebastiani. SAT-Based Bounded Model Checking for Timed Systems. In *Proc. FORTE'02.*, volume 2529 of *LNCS*. Springer, November 2002.

[ADB+18] Fatma Başak Aydemir, Fabiano Dalpiaz, Sjaak Brinkkemper, Paolo Giorgini, and John Mylopoulos. The Next Release Problem Revisited: A New Avenue for Goal Models. In *Proceedings of the 26th IEEE International Requirements Engineering Conference (RE'18)*, pages 5–16. IEEE, Aug 2018.

[ADR15] Mario Alviano, Carmine Dodaro, and Francesco Ricca. A maxsat algorithm using cardinality constraints of bounded size. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 2677–2683. AAAI Press, 2015.

[AGJ+14] Ozgur Akgun, Ian P. Gent, Christopher Jefferson, Ian Miguel, and Peter Nightingale. Breaking Conditional Symmetry in Automated Constraint Modelling with CONJURE. In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 3–8. IOS Press, 2014.

[ANOR13] Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. A Parametric Approach for Smaller and Better Encodings of Cardinality Constraints. In *19th International Conference on Principles and Practice of Constraint Programming*, CP'13, 2013.

[ANORC11] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality Networks: a theoretical and empirical study. *Constraints*, 16(2):195–221, 2011.

[Bal98] Egon Balas. Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89(1-3):3 – 44, 1998.

[Bar99] Roman Bartak. Constraint programming - what is behind? In *In Proceedings of the Workshop on Constraint Programming for Decision and Control (CPDC99*, pages 7–15, 1999.

[BB09] Robert Brummayer and Armin Biere. Boolector: An efficient smt solver for bit-vectors and arrays. In *Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems: Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009,*, TACAS '09, pages 174–177, Berlin, Heidelberg, 2009. Springer-Verlag.

[BBC+05a] M. Bozzano, R. Bruttomesso, A. Cimatti, A. Franzèn, Z. Hanna, Z. Khasidashvili, A. Palti, and R. Sebastiani. Encoding RTL Constructs for MathSAT: a Preliminary Report. In *Proc. 3rd Workshop of Pragmatics on Decision Procedure in Automated Reasoning, PDPAR'05*, ENTCS. Elsevier, 2005.

[BBC+05b] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Ranise, and R. Sebastiani. Efficient Satisfiability Modulo Theories via Delayed Theory Combination. In *Proc. CAV 2005*, volume 3576 of *LNCS*. Springer, 2005.

[BBC+05c] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi A. Junttila, Peter van Rossum, Stephan Schulz, and Roberto Sebastiani. Mathsat: Tight integration of sat and mathematical decision procedures. *Journal of Automated Reasoning*, 35(1-3):265–293, 2005.

[BBC+06] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi A. Junttila, Silvio Ranise, Peter van Rossum, and Roberto Sebastiani. Efficient Theory Combination via Boolean Search. *Information and Computation*, 204(10):1493–1525, 2006.

[BCF+06] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani. Delayed Theory Combination vs. Nelson-Oppen for Satisfiability Modulo Theories: A Comparative Analysis. In *Proc. LPAR*, volume 4246 of *LNCS*. Springer, 2006.

[BCF+07] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, Ziyad Hanna, Alexander Nadel, Amit Palti, and Roberto Sebastiani. A Lazy and Layered SMT($\mathcal{BV}$) Solver for Hard Industrial Verification Problems. In *CAV*, volume 4590 of *LNCS*, pages 547–560. Springer, 2007.

[BCI+14] Zhengbing Bian, Fabian Chudak, Robert Israel, Brad Lackey, William G Macready, and Aidan Roy. Discrete optimization using quantum annealing on sparse ising models. *Frontiers in Physics*, (56), 2014.

[BCM+17] Zhengbing Bian, Fabian Chudak, William Macready, Aidan Roy, Roberto Sebastiani, and Stefano Varotti. Solving SAT and MaxSAT with a Quantum Annealer: Foundations and a Preliminary Report. In *Frontiers of Combining Systems*, volume 10483 of *LNCS*, pages 153–171. Springer, 2017.

[BCM+18] Zhengbing Bian, Fabián A. Chudak, William G. Macready, Aidan Roy, Roberto Sebastiani, and Stefano Varotti. Solving SAT and maxsat with a quantum annealer: Foundations, encodings, and preliminary results. *CoRR*, abs/1811.02524, 2018. Under submission for journal publication.

[BD02] R. Brinkmann and R. Drechsler. RTL-datapath verification using integer linear programming. In *Proc. ASP-DAC 2002*, pages 741–746. IEEE, 2002.

[BDG+13] Martin Brain, Vijay D'Silva, Alberto Griggio, Leopold Haller, and Daniel Kroening. Interpolation-Based Verification of Floating-Point Programs with Abstract CDCL. In *SAS*, pages 412–432, 2013.

[BDG+14] Martin Brain, Vijay D'Silva, Alberto Griggio, Leopold Haller, and Daniel Kroening. Deciding floating-point logic with abstract conflict driven clause learning. *Formal Methods in System Design*, 45(2):213–245, 2014.

[BDS02] C. W. Barrett, D. L. Dill, and A. Stump. A generalization of Shostak's method for combining decision procedures. In *Frontiers of Combining Systems (FROCOS)*, LNAI. Springer-Verlag, April 2002. Santa Margherita Ligure, Italy.

[BdSR18] Clara Bertolissi, Daniel Ricardo dos Santos, and Silvio Ranise. Solving Multi-Objective Workflow Satisfiability Problems with Optimization Modulo Theories Techniques. In *SACMAT*, pages 117–128. ACM, 2018.

[BGH87] R. H. Byrd, A. J. Goldman, and M. Heller. Technical Note– Recognizing Unbounded Integer Programs. *Operations Research*, 35(1), 1987.

[BHvMW09] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*. IOS Press, February 2009.

[Bie18] Armin Biere. CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT Entering the SAT Competition 2018. In Marijn Heule, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2018 – Solver and Benchmark Descriptions*, volume B-2018-1 of *Department of Computer Science Series of Publications B*, pages 13–14. University of Helsinki, 2018.

[Bjo16] Nikolaj Bjorner. personal communication, 02 2016.

[BK11] Dirk Beyer and M. Erkan Keremoglu. Cpachecker: A tool for configurable software verification. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 184–190. Springer, 2011.

[BKW09] A. Brillout, D. Kroening, and T. Wahl. Mixed abstractions for floating-point arithmetic. In *2009 Formal Methods in Computer-Aided Design*, pages 69–76, Nov 2009.

[BLP+18] Arthur Bit-Monnot, Francesco Leofante, Luca Pulina, Erika Ábrahám, and Armando Tacchella. SMarTplan: a Task Planner for Smart Factories. *CoRR*, abs/1806.07135, 2018.

[BLPS15] Christina N. Burt, Nir Lipovetzky, Adrian R. Pearce, and Peter J. Stuckey. Scheduling with Fixed Maintenance, Shared Resources and Nonlinear Feedrate Constraints: A Mine Planning Case Study. In Michel [Mic15], pages 91–107.

[BNO+08] M. Bofill, R. Nieuwenhuis, A. Oliveras, E. Rodríguez Carbonell, and A. Rubio. The Barcelogic SMT Solver. In A. Gupta and S. Malik, editors, *20th International Conference on Computer Aided Verification, CAV'08*, volume 5123 of *Lecture Notes in Computer Science*, pages 294–298. Springer, 2008.

[BNOT06] C. Barrett, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Splitting on Demand in SAT Modulo Theories. In *Proc. LPAR'06*, volume 4246 of *LNAI*. Springer, 2006.

[BP14] Nikolaj Bjorner and Anh-Dung Phan. νZ - Maximal Satisfaction with Z3. In *Proc International Symposium on Symbolic Computation in Software Science*, Gammart, Tunisia, December 2014. EasyChair Proceedings in Computing (EPiC).

[BPF15] Nikolaj Bjorner, Anh-Dung Phan, and Lars Fleckenstein. Z3 - An Optimizing SMT Solver. In *Proc. TACAS*, volume 9035 of *LNCS*. Springer, 2015.

[BPSV09] Miquel Bofill, Miquel Palahı, Josep Suy, and Mateu Villaret. SIMPLY: a Compiler from a CSP Modeling Language to the SMT-LIB Format. In *Proceedings of the 8th International Workshop on Constraint Modelling and Reformulation*, pages 30–44, 2009.

[BPSV12] Miquel Bofill, Miquel Palahí, Josep Suy, and Mateu Villaret. Solving constraint satisfaction problems with SAT modulo theories. *Constraints*, 17(3):273–303, 2012.

[BPV09] Miquel Bofill, Miquel Palahı, and Mateu Villaret. A system for CSP solving through Satisfiability Modulo Theories. *IX Jornadas sobre Programación y Lenguajes (PROLE'09)*, pages 303–312, 2009.

[Bru09] Robert Brummayer. *Efficient SMT Solving for Bit-Vectors and the Extensional Theory of Arrays*. PhD thesis, Informatik, Johannes Kepler University Linz, 2009.

[BSST09] Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. *Satisfiability Modulo Theories*, chapter 26, pages 825–885. Volume 185 of Biere et al. [BHvMW09], February 2009.

[BSV10] Miquel Bofill, Josep Suy, and Mateu Villaret. A System for solving constraint satisfaction problems with SMT. *Theory and Applications of Satisfiability Testing–SAT 2010*, pages 300–305, 2010.

[BTRW15] Martin Brain, Cesare Tinelli, Philipp Rümmer, and Thomas Wahl. An Automatable Formal Semantics for IEEE-754 Floating-Point Arithmetic. In *ARITH*, pages 160–167. IEEE, 2015.

[CFG+10] Alessandro Cimatti, Anders Franzén, Alberto Griggio, Roberto Sebastiani, and Cristian Stenico. Satisfiability modulo the theory of costs: Foundations and applications. In *TACAS*, volume 6015 of *LNCS*, pages 99–113. Springer, 2010.

[CGI+17a] Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, Marco Roveri, and Roberto Sebastiani. *Invariant Checking of NRA Transition Systems via Incremental Reduction to LRA with EUF*, pages 58–75. Springer Berlin Heidelberg, Berlin, Heidelberg, 2017.

[CGI+17b] Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, Marco Roveri, and Roberto Sebastiani. Satisfiability Modulo Transcendental Functions via Incremental Linearization. In *Proc. Int. Conference on Automated Deduction, CADE-26*, LNCS. Springer, 2017.

[CGI+18] Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, Marco Roveri, and Roberto Sebastiani. Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. *ACM Trans. Comput. Log.*, 19(3):19:1–19:52, 2018.

[cgm] CGM-Tool. `http://www.cgm-tool.eu`.

[CGSS13a] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. A Modular Approach to MaxSAT Modulo Theories. In *International Conference on Theory and Applications of Satisfiability Testing, SAT*, volume 7962 of *LNCS*, July 2013.

[CGSS13b] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The MathSAT 5 SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS'13.*, volume 7795 of *LNCS*, pages 95–109. Springer, 2013.

[CLO+16] Lorenzo Candeago, Daniel Larraz, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. Speeding up the Constraint-Based Method in Difference Logic. In *SAT*, volume 9710 of *Lecture Notes in Computer Science*, pages 284–301. Springer, 2016.

[DAC10] M. Dib, R. Abdallah, and A. Caminada. Arc-consistency in constraint satisfaction problems: A survey. In *2010 Second International Conference on Computational Intelligence, Modelling and Simulation*, pages 291–296, Sep. 2010.

[DDA09] Isil Dillig, Thomas Dillig, and Alex Aiken. Cuts from Proofs: A Complete and Practical Technique for Solving Linear Inequalities over Integers. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification*, pages 233–247, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[DdM06a] B. Dutertre and L. de Moura. System Description: Yices 1.0. In *Proc. on 2nd SMT competition, SMT-COMP'06*, 2006. Available at `yices.csl.sri.com/yices-smtcomp06.pdf`.

[DdM06b] Bruno Dutertre and Leonardo de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *CAV*, volume 4144 of *LNCS*, 2006.

[DDMA12] Isil Dillig, Thomas Dillig, Kenneth L. McMillan, and Alex Aiken. Minimum Satisfying Assignments for SMT. In *CAV*, pages 394–409, 2012.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[DM06a] Bruno Dutertre and Leonardo De Moura. Integrating simplex with DPLL(T). Technical report, CSL, SRI INTERNATIONAL, 2006.

[DM06b] Bruno Dutertre and Leonardo De Moura. The yices smt solver. Technical report, SRI, 2006.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Model-based theory combination. *Electr. Notes Theor. Comput. Sci.*, 198(2):37–49, 2008.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

[DPSS15] Toby O. Davies, Adrian R. Pearce, Peter J. Stuckey, and Harald Søndergaard. Optimisation and Relaxation for Multiagent Planning in the Situation Calculus. In Gerhard Weiss, Pinar Yolum, Rafael H. Bordini, and Edith Elkind, editors, *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, pages 1141–1149. ACM, 2015.

[dwa] D-Wave 2X Tecnology Overview. Web page.

[EF14] Hani A Elgabou and Alan M Frisch. Encoding The Lexicographic Ordering Constraint in SAT Modulo Theories. In *In Proc. of Thirteenth International Workshop on Constraint Modelling and Reformulation*, pages 85–96, 09 2014.

[ES04] N. Eén and N. Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2919 of *LNCS*, pages 502–518. Springer, 2004.

[FBB18] Katalin Fazekas, Fahiem Bacchus, and Armin Biere. Implicit Hitting Set Algorithms for Maximum Satisfiability Modulo Theories. In *IJCAR*, volume 10900 of *Lecture Notes in Computer Science*, pages 134–151. Springer, 2018.

[FG10] Alan M. Frisch and Paul A. Giannaros. SAT Encodings of the At-Most-k Constraint Some Old , Some New , Some Fast , Some Slow. 2010.

[fla] FlatZinc 1.6. `http://www.minizinc.org/downloads/doc-1.6/flatzinc-spec.pdf`.

[FP14] Alan M Frisch and Miquel Palahí. Anomalies in SMT Solving: Difficulties in Modelling Combinatorial Problems. In *In Proc. of Thirteenth International Workshop on Constraint Modelling and Reformulation*, pages 97–110, 09 2014.

[FS09] Thibaut Feydy and Peter J. Stuckey. Lazy Clause Generation Reengineered. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming*, CP'09, pages 352–366, Berlin, Heidelberg, 2009. Springer-Verlag.

[fzn] FZN2SMT. `http://ima.udg.edu/Recerca/lap/fzn2smt/index.html`.

[Gas79] John Gary Gaschnig. *Performance Measurement and Analysis of Certain Search Algorithms.* PhD thesis, Pittsburgh, PA, USA, 1979. AAI7925014.

[GD07] Vijay Ganesh and David L. Dill. A Decision Procedure for Bit-Vectors and Arrays. In *CAV*, 2007.

[GKSS08] Carla P. Gomes, Henry Kautz, Ashish Sabharwal, and Bart Selman. *Satisfiability solvers*. Elsevier, 2008.

[Gri12] Alberto Griggio. A Practical Approach to Satisfiability Modulo Linear Integer Arithmetic. *Journal on Satisfiability, Boolean Modeling and Computation - JSAT*, 8:1–27, 2012.

[Had15] Liana Hadarean. *An Efficient and Trustworthy Theory Solver for Bit-vectors in Satisfiability Modulo Theories*. PhD thesis, New York University, 2015.

[HAMM14] Julien Henry, Mihail Asavoae, David Monniaux, and Claire Maïza. How to Compute Worst-case Execution Time by Optimization Modulo Theory and a Clever Encoding of Program Semantics. In *Proceedings of the 2014 SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems*, LCTES '14, pages 43–52, New York, NY, USA, 2014. ACM.

[HBJ+14] Liana Hadarean, Kshitij Bansal, Dejan Jovanovic, Clark Barrett, and Cesare Tinelli. A Tale of Two Solvers: Eager and Lazy Approaches to Bit-Vectors. In *CAV*, volume 8559 of *Lecture Notes in Computer Science*, pages 680–695. Springer, 2014.

[HGBK12] Leopold Haller, Alberto Griggio, Martin Brain, and Daniel Kroening. Deciding Floating-Point Logic with Systematic Abstraction. In *Proc. of FMCAD*, 2012. To Appear.

[IBM10] IBM. *IBM ILOG CPLEX Optimizer*, 2010. Available at `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`.

[iee08] IEEE standard 754, 2008. `http://grouper.ieee.org/groups/754/`.

[JAG+11] M. W. Johnson, M. H. S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, E. M. Chapple, C. Enderud, J. P. Hilton, K. Karimi, E. Ladizinsky, N. Ladizinsky, T. Oh, I. Perminov, C. Rich, M. C. Thom, E. Tolkacheva, C. J. S. Truncik, S. Uchaikin, J. Wang, B. Wilson, and G. Rose. Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198, 2011.

[JB10] Dejan Jovanovic and Clark Barrett. Polite theories revisited. In *LPAR (Yogyakarta)*, volume 6397 of *Lecture Notes in Computer Science*, pages 402–416. Springer, 2010.

[JG02] Jennifer R. Jackson and Ignacio E. Grossmann. High Level Optimization Model for the Retrofit Planning of Process Networks. *Ind. Eng. Chem. Res*, 41:41–3762, 2002.

[Kar17] George Egor Karpenkov. *Finding inductive invariants using satisfiability modulo theories and convex optimization*. Theses, Université Grenoble Alpes, March 2017.

[KBE17] Gergely Kovásznai, Csaba Biró, and Balázs Erdélyi. Generating Optimal Scheduling for Wireless Sensor Networks by Using Optimization Modulo Theories Solvers. 2017.

[KBE18] Gergely Kovásznai, Csaba Biró, and Balázs Erdélyi. Puli - a problem-specific omt solver. EasyChair Preprint no. 371, EasyChair, 2018.

[KBK09] Hans Kleine Büning and Oliver Kullmann. *Minimal Unsatisfiability and Autarkies*, chapter 11, pages 339–401. Volume 185 of Biere et al. [BHvMW09], February 2009.

[KEB18] Gergely Kovásznai, Balázs Erdélyi, and Csaba Biró. Investigations of graph properties in terms of wireless sensor network optimization. In *2018 IEEE International Conference on Future IoT Technologies (Future IoT)*, pages 1–8, Jan 2018.

[KFB16] Egor George Karpenkov, Karlheinz Friedberger, and Dirk Beyer. JavaSMT: A Unified Interface for SMT Solvers in Java. In *VSTTE*, volume 9971 of *Lecture Notes in Computer Science*, pages 139–148, 2016.

[Kha79] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244:1093–1096, 1979.

[Kul09] Oliver Kullmann, editor. *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*. Springer, 2009.

[Kum92] Vipin Kumar. Algorithms for constraint satisfaction problems: A survey. *AI MAGAZINE*, 13(1):32–44, 1992.

[LAK+14] Y. Li, A. Albarghouthi, Z. Kincad, A. Gurfinkel, and M. Chechik. Symbolic Optimization with SMT Solvers. In *POPL*, 2014.

[LÁN+17] F. Leofante, E. Ábrahám, T. Niemueller, G. Lakemeyer, and A. Tacchella. On the Synthesis of Guaranteed-Quality Plans for Robot Fleets in Logistics Scenarios via Optimization Modulo Theories. In *2017 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 403–410, Aug 2017.

[LAN+18] Francesco Leofante, Erika Abraham, Tim Niemueller, Gerhard Lakemeyer, and Armando Tacchella. Integrated Synthesis and Execution of Optimal Plans for Multi-Robot Systems in Logistics. *Information Systems Frontiers*, pages 1–21, May 2018.

[Lar02] Javier Larrosa. Node and arc consistency in weighted csp. In *Eighteenth National Conference on Artificial Intelligence*, pages 48–53, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[LLGCM10] Julien Legriel, Colas Le Guernic, Scott Cotton, and Oded Maler. Approximating the Pareto Front of Multi-criteria Optimization Problems. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *LNCS*, pages 69–83. Springer Berlin Heidelberg, 2010.

[LM09] Chu Min Li and Felip Manyà. *MaxSAT, Hard and Soft Constraints*, chapter 19, pages 613–631. Volume 185 of Biere et al. [BHvMW09], February 2009.

[lmt] LMT. `http://disi.unitn.it/~teso/lmt/lmt.tgz`.

[Lod09] Andrea Lodi. Mixed Integer Programming Computation. In *50 Years of Integer Programming 1958-2008*, pages 619–645. Springer-Verlag, 2009.

[LORR14] Daniel Larraz, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. Minimal-Model-Guided Approaches to Solving Polynomial Constraints and Extensions. In *SAT*, 2014.

[LTBT17] Tianhai Liu, Shmuel S. Tyszberowicz, Bernhard Beckert, and Mana Taghdiri. Computing Exact Loop Bounds for Bounded Program Verification. In *SETTA*, volume 10606 of *Lecture Notes in Computer Science*, pages 147–163. Springer, 2017.

[LTZ05] Marco Laumanns, Lothar Thiele, and Eckart Zitzler. An Adaptive Scheme to Generate the Pareto Front Based on the Epsilon-Constraint Method. In Jürgen Branke, Kalyanmoy Deb, Kaisa Miettinen, and Ralph E. Steuer, editors, *Practical Approaches to Multi-Objective Optimization*, number 04461 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[LW66] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Oper. Res.*, 14(4):699–719, August 1966.

[Mac77] Alan K. Mackworth. Consistency in networks of relations. *Artif. Intell.*, 8(1):99–118, February 1977.

[mata] MathSAT 5. `http://mathsat.fbk.eu/`.

[matb] MATHSAT5. `http://mathsat.fbk.eu`.

[Mic15] Laurent Michel, editor. *Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings*, volume 9075 of *LNCS*. Springer, 2015.

[mina] MiniZinc 1.6. `http://www.minizinc.org/downloads/doc-1.6/zinc-spec.pdf`.

[Minb] MiniZinc. `www.minizinc.org`.

[MJPL92] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: A heuristic repair method for constraint-satisfaction and scheduling problems. *ARTIFICIAL INTELLIGENCE*, 58(1):161–205, 1992.

[Mon74] Ugo Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Inf. Sci.*, 7:95–132, 01 1974.

[MP13] Panagiotis Manolios and Vasilis Papavasileiou. Ilp modulo theories. In *CAV*, pages 662–677, 2013.

[MSAGL11] João Marques-Silva, Josep Argelich, Ana Graça, and Inês Lynce. Boolean lexicographic optimization: algorithms & applications. *Ann. Math. Artif. Intell.*, 62(3-4):317–343, 2011.

[MSLM09] Joao P. Marques-Silva, Ines Lynce, and Sharad Malik. *Conflict-Driven Clause Learning SAT Solvers*, chapter 4, pages 131–153. Volume 185 of Biere et al. [BHvMW09], February 2009.

[MSP09] Vasco M. Manquinho, João P. Marques Silva, and Jordi Planes. Algorithms for weighted boolean optimization. In Kullmann [Kul09], pages 495–508.

[MsS99] João P. Marques-silva and Karem A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48:506–521, 1999.

[Nad14] Alexander Nadel. Bit-vector rewriting with automatic rule generation. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 663–679. Springer, 2014.

[NB14] Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided maxsat resolution. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2717–2723. AAAI Press, 2014.

[Nie17] Aina Niemetz. *Bit-Precise Reasoning Beyond Bit-Blasting*. PhD thesis, Informatik, Johannes Kepler University Linz, 2017.

[Nil80] N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1980.

[NO79] G. Nelson and D.C. Oppen. Simplification by Cooperating Decision Procedures. *ACM Trans. on Programming Languages and Systems*, 1(2):245–257, 1979.

[NO06] Robert Nieuwenhuis and Albert Oliveras. On SAT Modulo Theories and Optimization Problems. In *Proc. Theory and Applications of Satisfiability Testing - SAT 2006*, volume 4121 of *LNCS*. Springer, 2006.

[NORCR07] R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. Challenges in Satisfiability Modulo Theories. In *Proc. RTA'07*, volume 4533 of *LNCS*. Springer, 2007.

[NOT06] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, November 2006.

[NPFB15] Aina Niemetz, Mathias Preiner, Andreas Fröhlich, and Armin Biere. Improving Local Search For Bit-Vector Logics in SMT with Path Propagation. In *Proc. 4th Intl. Work. on Design and Implementation of Formal Tools and Systems (DIFTS'15)*, page 10 pages, 2015.

[NR16] Alexander Nadel and Vadim Ryvchin. Bit-Vector Optimization. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2016*, volume 9636 of *LNCS*. Springer, 2016.

[NSB+07] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a Standard CP Modelling Language. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, volume 4741 of *LNCS*, pages 529–543. Springer Berlin Heidelberg, 2007.

[NSGM16a] Chi Mai Nguyen, Roberto Sebastiani, Paolo Giorgini, and John Mylopoulos. Multi-objective reasoning with constrained goal models. *Requirements Engineering*, 2016. In print. Published online 24 December 2016. DOI: `http://dx.doi.org/10.1007/s00766-016-0263-5`.

[NSGM16b] Chi Mai Nguyen, Roberto Sebastiani, Paolo Giorgini, and John Mylopoulos. Requirements Evolution and Evolution Requirements with Constrained Goal Models. In *Proceedings of the 37nd International Conference on Conceptual Modeling - ER16*, LNCS. Springer, 2016.

[NSGM17] Chi Mai Nguyen, Roberto Sebastiani, Paolo Giorgini, and John Mylopoulos. Modeling and Reasoning on Requirements Evolution with Constrained Goal Models. In Alessandro Cimatti and Marjan Sirjani, editors, *Software Engineering and Formal Methods - 15th International Conference, SEFM 2017, Trento, Italy, September 4-8, 2017, Proceedings*, volume 10469 of *Lecture Notes in Computer Science*, pages 70–86. Springer, 2017.

[OCS18] R. Serna Oliver, S. S. Craciunas, and W. Steiner. IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 13–24, April 2018.

[Opp80] Derek C. Oppen. Complexity, convexity and combinations of theories. *Theoretical Computer Science*, 12(3):291 – 302, 1980.

[opt] OPTIMATHSAT. `http://optimathsat.disi.unitn.it`.

[OSC07] Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation = lazy clause generation. In *CP*, volume 4741 of *Lecture Notes in Computer Science*, pages 544–558. Springer, 2007.

[OSC09] Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009.

[Pap81] Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981.

[Pas16] Andrea Passerini. Learning Modulo Theories. In Christian Bessiere, Luc De Raedt, Lars Kotthoff, Siegfried Nijssen, Barry O'Sullivan, and Dino Pedreschi, editors, *Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach*, volume 10101 of *Lecture Notes in Computer Science*, pages 113–146. Springer, 2016.

[Pre09] Steven Prestwich. *CNF Encodings*, chapter 2, pages 75–97. Volume 185 of Biere et al. [BHvMW09], February 2009.

[Pug92] William Pugh. The Omega Test: a fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 8:4–13, 1992.

[pyl] PyLMT. `http://www.bitbucket.org/stefanoteso/pylmt`.

[Rat17] Stefan Ratschan. Simulation Based Computation of Certificates for Safety of Dynamical Systems. In Alessandro Abate and Gilles Geeraerts, editors, *Formal Modeling and Analysis of Timed Systems - 15th International Conference, FORMATS 2017, Berlin, Germany, September 5-7, 2017, Proceedings*, volume 10419 of *Lecture Notes in Computer Science*, pages 303–317. Springer, 2017.

[RBÅ18] S. F. Roselli, K. Bengtsson, and K. Åkesson. SMT Solvers for Job-Shop Scheduling Problems: Models Comparison and Performance Evaluation. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 547–552, Aug 2018.

[RBW06] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[REJ09] Derek Rayside, H.-Christian Estler, and Daniel Jackson. The Guided Improvement Algorithm for Exact, General-Purpose, Many-Objective Combinatorial Optimization. Technical report, Massachusetts Institute of Technology, Cambridge, 07 2009.

[RG94] R. Raman and I.E. Grossmann. Modelling and computational techniques for logic based integer programming. *Computers and Chemical Engineering*, 18(7):563 – 578, 1994.

[RM09] Olivier Roussel and Vaso Manquinho. *Pseudo-Boolean and Cardinality Constraints*, chapter 22, pages 695–733. Volume 185 of Biere et al. [BHvMW09], February 2009.

[Roc11] Oliver Vendrell Roc. Optimization Modulo Theories. Master's thesis, Polytechnic University of Catalonia, 2011. `http://hdl.handle.net/2099.1/14204`.

[RRZ05] Silvio Ranise, Christophe Ringeissen, and Calogero G. Zarba. Combining Data Structures with Nonstably Infinite Theories Using Many-Sorted Logic. In *FroCoS*, volume 3717 of *Lecture Notes in Computer Science*, pages 48–64. Springer, 2005.

[RW10] Philipp Ruemmer and Thomas Wahl. An SMT-LIB Theory of Binary Floating-Point Arithmetic. SMT 2010 Workshop, July 2010. Available at `http://www.philipp.ruemmer.org/publications/smt-fpa.pdf`.

[Sch99] Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

[Seb07] Roberto Sebastiani. Lazy Satisfiability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation, JSAT*, 3(3-4):141–224, 2007.

[SG05] Nicolas W. Sawaya and Ignacio E. Grossmann. A cutting plane method for solving linear generalized disjunctive programming problems. *Computing Chemical Engineering*, 29(9):1891–1913, 2005.

[SG12] Nicolas W. Sawaya and Ignacio E. Grossmann. A hierarchy of relaxations for linear generalized disjunctive programming. *European Journal of Operational Research*, 216(1):70–82, 2012.

[Sho84] R.E. Shostak. Deciding Combinations of Theories. *Journal of the ACM*, 31:1–12, 1984.

[Sin05] Carsten Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In Peter van Beek, editor, *CP*, volume 3709 of *LNCS*, pages 827–831. Springer, 2005.

[smt] SmtLibv2. `www.smtlib.cs.uiowa.edu/`.

[ST12] Roberto Sebastiani and Silvia Tomasi. Optimization in SMT with LA(Q) Cost Functions. In *IJCAR*, volume 7364 of *LNAI*, pages 484–498. Springer, July 2012.

[ST15a] Roberto Sebastiani and Silvia Tomasi. Optimization Modulo Theories with Linear Rational Costs. *ACM Transactions on Computational Logics*, 16(2), March 2015.

[ST15b] Roberto Sebastiani and Patrick Trentin. OptiMathSAT: A Tool for Optimization Modulo Theories. In *Proc. International Conference on Computer-Aided Verification, CAV 2015*, volume 9206 of *LNCS*. Springer, 2015.

[ST15c] Roberto Sebastiani and Patrick Trentin. Pushing the Envelope of Optimization Modulo Theories with Linear-Arithmetic Cost Functions. In *Proc. Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'15*, volume 9035 of *LNCS*. Springer, 2015.

[ST16] Roberto Sebastiani and Patrick Trentin. On the Benefits of Enhancing Optimization Modulo Theories with Sorting Networks for MaxSMT. In *Proceedings of the 14th International Workshop on Satisfiability Modulo Theories, SMT-2016.*, CEUR Workshop Proceedings, 2016.

[ST17] Roberto Sebastiani and Patrick Trentin. On Optimization Modulo Theories, MaxSMT and Sorting Networks. In *Proc. Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'17*, volume 10205 of *LNCS*. Springer, 2017.

[ST18] Roberto Sebastiani and Patrick Trentin. OptiMathSAT: A Tool for Optimization Modulo Theories. *Journal of Automated Reasoning*, Dec 2018.

[TCHS13] Sylvie Thiébaux, Carleton Coffrin, Hassan L. Hijazi, and John K. Slaney. Planning with MIP for Supply Restoration in Power Distribution Systems. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. IJCAI/AAAI, 2013.

[Tre14] Patrick Trentin. Linear Integer Optimization with SMT. Master's thesis, Università degli Studi di Trento, 2014. `https://pdfs.semanticscholar.org/263b/b026491dfdb6c6a87b4b5b9ec72f1a467047.pdf`.

[TS19] Patrick Trentin and Roberto Sebastiani. Optimization Modulo the Theory of Floating-Point Numbers. In *Proc. Int. Conference on Automated Deduction, CADE 27*, LNCS/LNAI. Springer, 2019. To appear.

[Tsa93] Edward Tsang. Foundations of Constraint Satisfaction, 1993.

[Tse68] G.S. Tseitin. *On the complexity of derivations in the propositional calculus*, page 115–125. Consultants Bureau, 1968. Part II.

[TSP17] Stefano Teso, Roberto Sebastiani, and Andrea Passerini. Structured learning modulo theories. *Artificial Intelligence*, 244:166–187, 2017.

[TZ05] Cesare Tinelli and Calogero G. Zarba. Combining Nonstably Infinite Theories. *J. Autom. Reasoning*, 34(3):209–238, 2005.

[vD94] Dirk van Dalen. *Logic and structure (3. ed.)*. Universitext. Springer, 1994.

[VG04] A. Vecchietti and I.E. Grossmann. Computational experience with logmip solving linear and nonlinear disjunctive programming problems. In *Proc. of FOCAPD*, pages 587–590, 2004.

[VL01] Axel Van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *Proceedings of the Fifth IEEE International Conference on Requirements Engineering*, RE'01, pages 249–. IEEE Computer Society, 2001.

[VS10] Michael Veksler and Ofer Strichman. A proof-producing CSP solver. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

[VS15] Michael Veksler and Ofer Strichman. Learning general constraints in CSP. In Michel [Mic15], pages 410–426.

[VS16] Michael Veksler and Ofer Strichman. Learning general constraints in CSP. *Artif. Intell.*, 238:135–153, 2016.

[Wal75] David Waltz. Understanding line drawings of scenes with shadows. In *The Psychology of Computer Vision*, page pages. McGraw-Hill, 1975.

[wce] WCET OMT. `https://github.com/PatrickTrentin88/wcet_omt`.

[z3] Z3. http://research.microsoft.com/en-us/um/redmond/projects/z3/ml/z3.html.

[ZBWR18] Aleksandar Zeljić, Peter Backeman, Christoph M. Wintersteiger, and Philipp Rümmer. Exploring approximations for floating-point arithmetic using uppsat. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning*, pages 246–262, Cham, 2018. Springer International Publishing.

[ZK17] Neng-Fa Zhou and Hakan Kjellerstrand. Optimizing SAT Encodings for Arithmetic Constraints. pages 671–686, 08 2017.

[ZMMM01] Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz, and Sharad Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of the 2001 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '01, pages 279–285, Piscataway, NJ, USA, 2001. IEEE Press.

[ZWR14] Aleksandar Zeljić, Christoph M. Wintersteiger, and Philipp Rümmer. Approximations for model construction. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *Automated Reasoning*, pages 344–359, Cham, 2014. Springer International Publishing.

[ZWR17] Aleksandar Zeljić, Christoph M. Wintersteiger, and Philipp Rümmer. An approximation framework for solvers and decision procedures. *Journal of Automated Reasoning*, 58(1):127–147, Jan 2017.